CramSession
The Original Study Guide

Over 3 Million Downloaded

Sun
Solaris 9 Security
Sun Certified Security Administrator for the Solaris 9 Operating Environment

CX-310-301 **CX-310-301** CX-310-301 CX-310-301

Written by **Subject Matter Experts**

Your Trusted Study Resource for **Technical Certification**

The **Most Popular Study Guide** on the web

Introduction

This CramSession will help you prepare for the Solaris 9 Sun Certified Security Administrator. The exam topics that are covered by this document include general security concepts, device management and detection of devices, the types of security attacks that can be used, the protection of file and system resources, preventing attacks on hosts and networks and how to protect network access using encryption and authentication.

It is recommended by Sun, although not compulsory that you have worked in a security administration role for at least six months and have attained the Solaris system administration and network administration certifications.

About the Solaris 9 Operating System Exam

Sun's exam CX-310-301, "Sun Certified Security Administrator for the Solaris 9 Operating Environment" is the only requirement for the Sun Certified Security Administrator for the Solaris Operating Environment certification. It is designed for candidates with "six to twelve months security administration job-role experience" and who have a previous Solaris OE and network administration certification. Although Sun does not list a "Sun Certified System Administrator" (SCSA) certification or a "Sun Certified Network Administrator" (SCNA) certification as prerequisites, these would be logical certifications to hold before taking on this security certification.

This exam costs $150 USD and may be taken from Thomson Prometric.

The pass score for this multiple choice, drag and drop, and matching exam is 60%. There are 60 questions and the exam lasts 90 minutes.

The topics covered by this exam include:

- General Security Concepts
- Detection and Device Management
- Security Attacks
- File and System Resources Protection
- Host and Network Prevention
- Network Connection Access, Authentication, and Encryption

For more information:
http://training.sun.com/US/catalog/courses/CX-310-301.html

Exam Time Tips

It is highly recommended that you have access to at least two SPARC workstations with Solaris 9 installed. This will allow you to try out all of the methods shown in this document and to become familiar with how they work and the potential problems that can be encountered. The workstations need to be networked to allow remote communications to be allowed/blocked.

Make use of the manual pages because they provide a wealth of information about the utilities as well as full descriptions of the syntax and available options. For packages that are installed, ensure you have the directory */usr/local/man* added to the *MANPATH* variable so these are accessible as well.

Use Sun's website for documentation and "how-to" guides. They are comprehensive guides and will provide expert information and guidance.

Additional Resources

CramSession Solaris 9 Security Certification Page

CramSession Certification & Training Solaris 9 Security Articles

Solaris 9 Security Feedback and Discussion Board

Free Question of the Day

CRAMSESSION™ SINGLE USER LICENSE

This is a legal agreement between you, an individual user, and CramSession. CramSession provides you with the content, information, and other material associated with this CramSession™ study guide, hereinafter referred to as the "Content," solely under the following terms and conditions, hereinafter referred to as the "License." By accessing the Content, you agree to be bound by the terms of this License. If you do not agree to be bound by the terms of this License, do not access, view, or use the Content. Each time you access, view, or use the Content, you reaffirm your agreement to and understanding of this License.

1. Grant of License.

CramSession hereby grants to you a nonexclusive, nontransferable, non-assignable, limited right and license to access, view, and use the Content on one (1) computer or workstation at a time for your individual, personal, non-commercial use. You may further print one (1) copy of the Content for your individual, personal, non-commercial use, but may not otherwise copy or reproduce the Content.

You may not share or allow others to view the Content. You may not network the Content or display or use it on more than one computer or workstation at the same time. You may not upload, post, transmit, or distribute printed or electronic versions of the Content in any manner to any other party. You may not sell, rent, lease, lend, sublicense, or otherwise transfer or distribute the Content to any other party. You may not modify or create a derivative work based on the Content. You may not modify or delete any proprietary notices contained in the Content, including, but not limited to, any product identification, product restriction, trademark, copyright, or other proprietary notices.

2. Term of License.

In the event that you are in breach of any provision of this License, this License shall thereby be automatically terminated with no further action required by CramSession. In the event of such termination, you agree to immediately destroy all printed and electronic versions of the Content in your possession, custody, or control.

3. Ownership.

The Content is the proprietary product of CramSession and is protected by copyright, trade secret, and other intellectual property laws. You are acquiring only the right to access, view, and use the Content as expressly provided above. CramSession now holds and shall retain all right, title, and interest in and to the Content, including, but not limited to, all copyrights, patent rights, trade secret rights, trademark rights, and other similar property rights with respect to the Content. Upon termination of this License, you shall retain no rights of any nature with respect to the Content.

4. Limited Warranty and Limited Liability.

YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT USE OF THE CONTENT IS AT YOUR OWN RISK AND THAT CRAMSESSION PROVIDES, AND YOU ACCEPT, THE CONTENT "AS IS" WITHOUT ANY WARRANTIES, CONDITIONS, OR REPRESENTATIONS WHATSOEVER; AND CRAMSESSION DISCLAIMS ANY AND ALL WARRANTIES, CONDITIONS, AND REPRESENTATIONS (STATUTORY, EXPRESS, OR IMPLIED, ORAL OR WRITTEN), WITH RESPECT TO THE CONTENT, INCLUDING, BUT NOT LIMITED TO, ANY AND ALL IMPLIED WARRANTIES OR CONDITIONS OF TITLE, NONINFRINGEMENT, MERCHANTABILITY, ACCURACY OR FITNESS OR SUITABILITY FOR ANY PARTICULAR PURPOSE (WHETHER OR NOT CRAMSESSION KNOWS, HAS REASON TO KNOW, HAS BEEN ADVISED, OR IS OTHERWISE IN FACT AWARE OF ANY SUCH PURPOSE), WHETHER ALLEGED TO ARISE BY LAW, BY REASON OF CUSTOM OR USAGE IN THE TRADE, OR BY COURSE OF DEALING. CRAMSESSION DOES NOT WARRANT THAT THE CONTENT WILL MEET YOUR REQUIREMENTS. SOME STATES OR COUNTRIES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO CERTAIN OF THE ABOVE EXCLUSIONS MAY NOT APPLY.

CRAMSESSION SHALL NOT BE LIABLE TO YOU UNDER ANY CIRCUMSTANCES FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR PUNITIVE DAMAGES, OR DAMAGES FOR LOSS OF PROFITS, REVENUE, DATA, OR DATA USE, INCURRED BY YOU ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE CONTENT, WHETHER IN AN ACTION IN CONTRACT OR TORT, EVEN IF CRAMSESSION HAS BEEN ADVISED OR IS AWARE OF THE POSSIBILITY OF SUCH DAMAGES. CRAMSESSION'S LIABILITY FOR DAMAGES SHALL IN NO EVENT EXCEED THE FEES PAID BY YOU FOR THIS LICENSE.

The Content may be subject to export restrictions. You agree that you will not export the Content or any part thereof to any country, person, entity, or end user subject to U.S. export restrictions. You expressly agree not to export any part of the Content to any country to which the U.S. has embargoed or restricted the export of goods or services, or to any national who intends to export the Content back to any embargoed country. You warrant and represent that no U.S. federal agency has suspended, revoked, or denied your export privileges.

CramSession may use fictitious names of companies, products, and individuals in the Content. These fictitious names are not intended to represent any real companies, products, or individuals.

CramSession may use real product names in the Content for informational purposes only. By using these product names, CramSession does not imply any endorsement of or affiliation with these products or their manufacturers. Some product names may be registered trademarks of their owners.

The Content contains links that may take you to other third-party web sites, pages, or services not under CramSession's control. CramSession provides these links on its web site only as a convenience to you. CramSession is not responsible for content of these third party offerings. You should not infer that CramSession endorses the content accessible through these links in any way.

5. Applicable Law.

This agreement shall be governed in its construction, interpretation, and performance by the laws of the State of Florida, without reference to law pertaining to choice of laws or conflict of laws. In the event of any claim or dispute arising out of or relating to this agreement or the breach, termination, validity, or enforcement of this agreement, venue shall be exclusively in Pinellas County, Florida.

PLEASE READ CAREFULLY. THE FOLLOWING LIMITS SOME OF YOUR RIGHTS, INCLUDING THE RIGHT TO BRING A LAWSUIT IN COURT. By accepting this agreement, you and CramSession agree that all claims or disputes between us will be submitted to binding arbitration if demanded by either party. The arbitration will be handled by the American Arbitration Association and governed by its rules. This agreement requiring arbitration (if demanded) is still fully binding even if a class action is filed in which you would be a class representative or member. You and CramSession agree that the arbitration of any claim or dispute between the parties will be conducted apart from all other claims or disputes of other parties and that there will be no class or consolidated arbitration of any claim or dispute covered by this agreement. You and CramSession also agree that this agreement does not affect the applicability of any statute of limitations.

6. Waiver.

No failure or delay on the part of CramSession in exercising any right or remedy with respect to a breach of this agreement by you shall operate as a waiver thereof or of any prior or subsequent breach of this agreement by you, nor shall the exercise of any such right or remedy preclude any other or future exercise thereof or exercise of any other right or remedy in connection with this agreement. Any waiver must be in writing and signed by CramSession.

Security Administrator for the Solaris 9 Operating System

General Security Concepts

This section is concerned with describing a number of fundamental concepts and terms relating to computer and information security. It also covers an analysis of a potential attacker, how crucial information can be gained by ill-prepared security procedures and the motives and methods of an attacker.

Information Security

Information security is not just about the security of data, it encompasses three main objectives:

- Protection of physical access to a computer system or network
- Controlling authorized access to a computer system or network through the use of user accounts and passwords
- Protection of the data and database information from unauthorized access, which includes accidental, as well as deliberate, deletion or modification.

In order to achieve "information security" all three of the above must be addressed equally. Failure in any one aspect can nullify the effect of the others. For example, if an attacker is allowed to gain physical access to a system console, then the data and user accounts are put at risk.

The Security Life Cycle

An extremely popular misconception is that security is a task that has to be carried out once and having carried it out, the systems are secure, allowing everyone to go and do something more interesting. This could not be more wrong. Computer security is an ongoing task that needs to be revisited regularly to reduce the chances of a security lapse. Periodic maintenance of the security procedures ensure that the security of the systems and data remains effective. With Solaris systems, this includes the application of regular security and recommended patches, effective management of user accounts and testing for vulnerabilities.

A security life cycle helps you to understand and formulate procedures for keeping the security procedures current. It identifies four distinct phases:

- **Prevent** – The initial hardening of a system (or network) where services that might not be required are turned off. A good baseline is to turn off all the services and then determine the ones you really need – this helps the administrator to be sure that no services are missed. Additionally, when a new service is requested, any risks associated with the service should be considered BEFORE it is enabled.

- **Detect** – You should, at regular intervals, run tests to see if you can break in to your systems. A number of scanning tools and vulnerability checking applications are available to do this. The value of this is that you can detect new vulnerabilities at the earliest opportunity – remember, if you find a vulnerability, then it's a good bet an attacker already knows about it! As well as testing for new kinds of threat, you can also run a system audit – this can be extremely useful, for example, to detect changes to files that should not have changed, thereby indicating a breach of security.
- **React** – If a security breach is located at the detection phase, then you need to react to it and "plug the hole". This might involve applying a patch to remedy the situation, but a review of services and applications should also be carried out to see if they are all still required. Attackers thrive on legacy systems and applications that are "just left running" – often a legacy application has already been migrated to another system, but no-one has turned the old one off!
- **Deter** – You can't protect against everything, or foresee what is going to happen, but you can take reasonable precautions to ensure your systems and applications are not left wide open for an attacker to exploit. If a system or application is not needed, shut it down immediately! It is no longer a risk. Another useful deterrent is to place a notice on the entrance to a system – for example when a user logs in. It won't stop a determined attacker, but should state that there should be no unauthorized access. Attackers have been cleared of crimes in the past because there was nothing saying that they shouldn't be logged in!

The most important point to remember about the security life cycle is that it is a cycle and not a one-off implementation.

Good Security

There are always risks involved in computer security and you can never be 100% certain that you won't be attacked or compromised. Even disconnecting your systems from the rest of the world doesn't protect you from an attack from the inside. The following factors are critical in achieving good security:

- **The people** – Education of users and training are probably the most important aspect because a user that is aware of the risks and takes security seriously is a precious asset.
- **Processes** – Regular procedures to check the system and application security need to be carried out to ensure the effectiveness of the security policy that is implemented at your site. Many companies decide that this should be done, but how many actually do it? And are then surprised when a vulnerability is exploited by an attacker.
- **Technology** – Apply the patches regularly to your systems. Sun Microsystems tend to update the recommended patch cluster about twice a month and can be downloaded directly from their web site. Read newsgroups and see what other users are experiencing – useful information can often be gathered from these locations. Run an intrusion detection system (IDS) so that you can be alerted to probes or attacks from other computers or networks.
- **Defense in depth** – Consider applying different protection mechanisms at several layers on your systems. The more protection you have, the harder it is for an attacker to penetrate and cause serious damage. If an attacker has to penetrate a number of obstacles before gaining access, then it

is less likely that he/she will continue with the attack. Compare this aspect with adding security to your motor car – alarms, immobilizers, steering wheel clamps, wheel clamps and so on. This creates problems for the attacker (or thief) and will take longer to break in. Remember, there is always another, easier opportunity for the attacker, just make sure it isn't you! A simple example is allowing the root user to login only from the system console. This means a potential attacker must penetrate two user accounts before gaining privileged access to the system.

Security Awareness

Being aware that security is an issue does not constitute security awareness. Security awareness is the understanding that computer security involves a number of aspects at different levels and that all the levels collectively provide the security that is required.

Security Policies

A security policy is an unambiguous document that describes the framework for protecting the company's assets and staff. It defines what is permitted and what is not permitted as well as any tolerances. An important aspect of a security policy is that it should clearly state one of two assumptions – either everything that is not explicitly permitted will be denied or that everything that is not explicitly denied will be permitted – the former is the normal course of action.

A security policy reflects the specific security requirements of a company and should detail not only what the policy covers, but also what it excludes. It must be explicit in the systems, buildings, networks, people and media that are being protected by the policy and why they are important as well as how to protect them. Another important aspect of a security policy is the procedure to follow if a security breach occurs. A lot of policies merely lay down a number of rules, but do not detail what happens if the rules are broken!

Every security policy should address the following topics in detail:

Policies and Procedures

Every company should have a security policy describing the rules for protecting the staff and assets. The policy was defined above. Here, the procedures are examined. A security policy needs to contain the following information:

- Which assets are covered by the policy
- The reason for the assets to be protected
- Who is responsible for each asset
- How the asset is physically accessed
- The threats and risks to the asset
- Password selection criteria
- The applications and services that are available to be utilized and those that are not allowed (Internet chat rooms, games, download sites for example)

- The procedure to follow in the event of a security breach
- Any special dispensation procedures, for example, to allow rapid deployment of a system or application before being fully accredited to the policy
- References to Data Protection legislation and how the policy complies with the legal requirements

Physical Security

This describes the physical security measures that must be taken to protect the assets described in the policy and must include the following:

- Location of the asset
- Access to the asset during normal working hours and, if access is permitted out of hours, what special measures are taken
- Emergency procedures
- Any special access methods, such as swipe cards, keys and so on
- Any theft prevention methods, such as asset tagging, secure fixings and so on

Platform Security

Platform security relates to the entire platform (PC or Unix for example) and details the procedures that must be followed to implement a server for the designated platform. Of particular interest is the use of any authentication modules that need to be applied, or the delegation of administrator functions to other user accounts (roles and *sudo* for example).

Network Security

This aspect is primarily concerned with protecting the company's data whilst it is being transferred from one system to another, i.e. on the network. The network security section should include details of network protection mechanisms and devices:

- Firewalls
- Virtual Private Networks (VPN)
- Routers
- Encryption methods used
- Any intrusion prevention mechanisms used
- Any authentication mechanisms used, such as single sign-on applications like Sun Enterprise Authentication Mechanism (SEAM)

Application Security

An insecure application can undermine the entire security policy and must be treated with respect when defining a security policy. Most of the time, you will not have the source code for an application (unless it is open source), so there is a reliance on the supplier to provide a "fix" for a security problem. The security policy must state the accepted tolerance (if any) to allow a solution to be implemented. In extreme circumstances, the policy could state that the application must not be used until a documented solution is installed and tested.

Security Operations and Management

The following recommendations will assist the smooth running of a security policy:

- If possible, install the latest release of the operating environment. Sun Microsystems release an update approximately every 3 months. Some of these will contain new security features or auditing facilities.
- Ensure that you have applied the most recent patches on a regular basis and look for security patches as a priority
- Establish a "fingerprint" database so it is easy to determine if a file or program has been tampered with. Sun Microsystems provide the Sun Fingerprint database for this purpose, but Tripwire (www.tipwire.com) carries out a similar function.
- Ensure that sufficient logging and auditing is enabled on the system to be able to catch important messages and events
- Identify a sensible backup strategy, stick to it and carry out periodic tests of backup media. This facility is essential if a security breach occurs and the system needs to be restored to a time prior to the attack.
- Adhere to a realistic user account management procedure. Run a password cracking program to identify weak passwords and disable dormant user accounts.
- Ensure that the user community is educated and informed of potential security risks and that the user is aware of security issues.
- Never allow a compiler to exist on a production system. This is an attacker's dream – you might as well leave the *root* password on your company's homepage!

Insecure Systems

An insecure system is one that is not adequately protected against unauthorized access. This could also be the result of poor internal security allowing an authorized user of the system to gain access to privileged information or utilities. Reasons for insecure systems include:

- **Lax file/directory permissions**
- Poor user account management – allowing users to login without a password, or using a weak password

- Unnecessary services and ports being available, allowing known vulnerabilities to be exploited
- The system giving out too much information to potential attackers
- No firewall implemented
- No logging of failed login attempts, which would indicate, for example, an attacker trying to guess passwords
- No auditing of operations, such as file deletions

User Trust

With any computer network, or computer system, there has to be an element of trust between the system administrator and the user community. An employee, for example, is trusted to a certain extent because he/she is working for the company. Also, effective computer security involves balancing the need for security with permitting people to do their job. An over cautious security regime can have a dramatic effect on productivity because it takes much longer to do even the simplest task.

Threat

A threat is something that could be deemed a potential target to an attacker. A good security policy should identify the relevant threats to an organization, assess the likelihood of an attack being successful and also establish the damage that could be caused (usually in financial terms).

Risk

There is always a chance that something might happen. The term "risk" is associated with the likelihood of an event occurring, coupled with the impact that it would have if it did occur. In assessing a risk to an asset, the outcome is normally one of four options:

- **Acceptance** – The cost of dealing with the risk might be too great, and the chances of it happening quite low. The risk can be accepted because nothing can be done to prevent it.
- **Avoidance** – Action can be taken to completely remove the risk
- **Reduction** – Action can be taken to reduce the chances of the event happening, or at least its effect is minimized.
- **Transfer** – Move the risk to another system, which might be much more secure from external attackers. Sometimes this option can also involve taking out insurance, especially where the risk might be related to hardware theft.

Authentication and Privacy

This subsection describes a number of terms and concepts:

Accountability

Accountability is the assignment of responsibility, frequently associated with user accounts on computer systems. When you, as a user, are given a user account and password, you become accountable (responsible) for all actions carried out by that user. Shared user accounts that are used by more than one person undermine the accountability – how can you be certain of who did what? Maintaining accountability is an important aspect of computer security. Companies often implement accountability on the assumption that "if it's your user account, you're responsible". It means that if you give your password to someone else and that person causes untold damage, you will be held "accountable".

Authentication

Authentication is the ability to prove who you are, i.e. your identity. It is not limited to human beings, it might be a computer program accessing another, remote system. Authentication can be proved in a number of ways:

- By entering a password
- By entering a pass phrase, used in secure communications
- By swiping a smartcard
- By IP address recognition
- By a trusted digital certificate from a trusted agent, such as Verisign.

Authorization

Authorization occurs after authentication and is the check that the user or system possesses the correct rights to be able to access an asset, such as a data file or database.

Authorizations are provided (and restricted) through any of the following:

- Granting of permissions (*chmod*)
- Granting and revoking of database privileges
- Adding a user to a group
- Assigning a role to a user
- Using Access Control Lists (ACL)

Privacy

This is an important aspect as it has legal implications. A lot of countries implement a data protection act and it is the holder of the information's responsibility to protect personal and private data that might be held, such as credit card information, names and addresses and so on. Privacy normally relates to sensitive or personal information. The privacy of data can be targeted by a potential attacker for two main reasons:

- **Individual** – Detailed information on an individual person, family, company or Government is targeted. Terrorists and criminals might use this approach.
- **Data Harvest** – Bulk data is targeted normally by criminal elements for the purpose of a scam. This might include personal, or financial information so that groups of people can be targeted automatically. A good, implemented security policy with data encryption can offer much greater protection from this type of privacy violation.

Confidentiality

Very similar to the definition of privacy, but confidentiality is concerned with preventing unauthorized disclosure of information. Confidential information is normally data which could be used by others to gain advantage and differs from private information in that it might not be personal in nature, or subject to any data protection legislation.

Integrity

Privacy and confidentiality are concerned with not letting unauthorized persons or systems read data, whereas integrity is concerned with the data itself and it's known condition. The integrity of any data is that it must be in the same state and condition as when it was last written by an authorized person or process and that it has not been altered for example by a computer virus or a disk error or a malicious attacker.

Non-repudiation

This is the evidence that something took place, making it impossible to deny. For example, being able to prove that an email message was sent and delivered, similar to a recorded postal delivery requiring the recipient to "sign for" the goods. In this case, it is very difficult (if not impossible) to deny that it was delivered.

Attackers

This section describes the types of attackers and why you might be attacked. It also discusses how attackers obtain vast amounts of information in support of their illicit activities.

Classification of Attackers and Motives

- **Script Kiddies** – These are amateurs who have little or no experience of breaking in to computer systems. They do it for fun mainly or the kudos of saying that they broke in. A script kiddie will normally run a program or utility supplied to him and won't understand the underlying security implications.
- **Hackers** – More experienced computer users and programmers that break in to computers and networks, but cause little or no damage. They are still trespassing however, but are often differentiated from the cracker, who will often cause malicious damage to systems or data.

- **Employees** – Probably the worst form of attacker is one from within. Normally an employee with a grudge against the company – no pay rise, no prospects, recently missed out on a promotion for example. The internal employee knows the business and can potentially cause untold damage. The majority of attacks still come from within.
- **Criminals** – Individuals are not normally very experienced and are looking for ways to make "easy money". Organized gangs are completely different and will use extortion methods as well as industrial espionage. A good example is the blackmailing of online bookmakers where they are threatened with Denial of Service attacks during premium sporting events unless money is paid.
- **Terrorists** – Often highly organized, but not concerned with covering their tracks (like a hacker). Attacks are carried out to further their cause and can be ruthless. Terrorists normally attack a specific target unlike others who will be looking for any decent opportunity. Frequently terrorists will undertake attacks on websites to gain publicity – defacing home pages is a popular method used to spread propaganda messages.
- **Natural Causes** – A frequently overlooked type of attack because it is not deliberate in its nature. The accidental deletion of important files is a good example where good security would prevent a user from being able to carry out the operation. Also in this aspect, consider such things as earthquake, flood and fire.

Information Gathering

An important point about information gathering is that you have to view your systems from the perspective of the attacker in order to understand how best to protect the assets you are responsible for. This section looks at the type of information an attacker is looking for and how specific information can be elicited from systems and people. Also this section discusses ways in which to combat these techniques.

Footprinting and fingerprinting are two major techniques used by an attacker in the information gathering phase of planning an attack. By utilizing these techniques on your own system, you can pre-empt the majority of ways in which an attacker might gain useful information and close them, if possible.

- **Footprinting** – Researching the target, probing websites, looking for information on the company using online search engines, newsgroups etc. A lot of information is often available about companies through these avenues. Also finding out domain name information, details about IP addresses (or blocks of IP addresses), telephone numbers – perhaps the company has reserved a block of telephone numbers, email addresses which could prove useful for virus proliferation, and where the offices are located. The location information could prove extremely useful because this opens up the opportunity to pose as an employee from another site in order to gain restricted or confidential information.
- **Fingerprinting** – This is the next phase of information gathering where you would run port scanners and network probes against the company network, looking for potential vulnerabilities that could be exploited in order to gain access. Details about the operating systems, services, whether there is a firewall installed, whether the systems are reachable across the Internet are all examples of valuable information that can be gathered. Having gained operating system

information for example, the attacker would consult publicly available sites such as www.cert.org or www.sans.org for news on vulnerabilities.

Attackers choose their targets based on a variety of criteria, depending on the overall objective. A terrorist will target a specific company or type of company, whereas a cracker will just scan around looking for a vulnerable site to break into and cause damage.

Gathering Techniques

Attackers use three main techniques for gathering information:

- **Social Engineering** – This technique takes advantage of human nature and is used to unintentionally reveal vital information. This includes:
    - **Shoulder Surfing** – Looking over a user's shoulder as a password is entered for example.
    - **Helpdesk Call** – Logging an urgent call with the helpdesk, posing as another user and getting the password reset for example.
    - **Post-It Notes** – It is amazing how much sensitive information can be found on notice boards or people's desks.
    - **Email Deception** – This is where an attacker sends a user an email requesting authentication (username and password) before the user can continue.

- **Technical Engineering** – Hosts that are connected to a network, particularly the Internet, have to give out a certain amount of information. Most though, give out far too much. An attacker can exploit this to gain technical information about the system and other systems connected to it on the local network. These include:
    - **ping** – Using the *ping* command with various options tests not only the reachability of a system, but also other information, such as the route taken to get to the system and the IP address of the hostname.
    - **traceroute** – Using this utility traces the precise route to the target system, identifying all the routers on the way.
    - **rpcinfo** – The *rpcinfo* command provides details of a remote host and the RPC services it is running, as shown here when using the *–p* option, it reveals a large amount of information:

```
# rpcinfo -p ultra10

   program  vers proto    port  service
    100000    4   tcp      111   rpcbind
    100000    3   tcp      111   rpcbind
```

```
    100000    2   tcp      111  rpcbind
    100000    4   udp      111  rpcbind
    100000    3   udp      111  rpcbind
    100000    2   udp      111  rpcbind
    100024    1   udp    32772  status
    100024    1   tcp    32771  status
    100133    1   udp    32772
    100133    1   tcp    32771
    100021    1   udp     4045  nlockmgr
    100021    2   udp     4045  nlockmgr
    100021    3   udp     4045  nlockmgr
    100021    4   udp     4045  nlockmgr
    100021    1   tcp     4045  nlockmgr
    100021    2   tcp     4045  nlockmgr
    100021    3   tcp     4045  nlockmgr
    100021    4   tcp     4045  nlockmgr
    100005    1   udp    32803  mountd
    100005    2   udp    32803  mountd
    100005    3   udp    32803  mountd
    100005    1   tcp    32776  mountd
    100005    2   tcp    32776  mountd
    100005    3   tcp    32776  mountd
    100003    2   udp     2049  nfs
    100003    3   udp     2049  nfs
    100227    2   udp     2049  nfs_acl
    100227    3   udp     2049  nfs_acl
    100003    2   tcp     2049  nfs
    100003    3   tcp     2049  nfs
    100227    2   tcp     2049  nfs_acl
    100227    3   tcp     2049  nfs_acl
    300598    1   udp    32805
    300598    1   tcp    32778
 805306368  1     udp    32805
 805306368  1     tcp    32778
    100249    1   udp    32806
    100249    1   tcp    32779
```

- **telnet and ftp** - The *telnet* command to the mail port (port 25) can be used to find out valid user accounts on a system. The example below shows a session to host 0, or localhost, and queries a number of users. Notice that when a valid user account is found, the full name and email address details are returned:

```
# telnet 0 25
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
```

```
220 ultra10.example.com ESMTP Sendmail 8.12.10+Sun/8.12.9; Thu, 1 Apr
2004 18:38:49 +0100 (BST)
expn john
250 2.1.5 John Philcox <john@ultra10.example.com>
expn testuser
550 5.1.1 testuser... User unknown
expn admin
250 2.1.5 System Administrator <admin@ultra10.example.com>
quit
221 2.0.0 ultra10.example.com closing connection
Connection to 0 closed by foreign host.
```

Steps to Prevent Information Disclosure

There are a number of steps that can be taken to avoid information disclosure:

- User education – The most important step that can be taken. Make people aware of the security issues and methods by which attackers might try and obtain information from them that would compromise the computer system
- Turn off any unnecessary network services
- Enforce a good password mechanism and educate users again to realize the importance of choosing a secure password
- Use encryption mechanisms for data in transit between systems, especially on the Internet
- Implement a firewall to protect the organization and add logging to the rules so that a record exists of persistent attempts to access an asset
- Enable auditing and system logging so that unauthorized attempts are recorded and the accountability and non-repudiation is maintained
- Monitor the systems continuously, which might mean installing a third-party product so that the administrator is automatically informed when events occur – products such as Sun Management Center, HP Openview, BMC Patrol, eTrust from Computer Associates are good examples

Evaluation Standards

The National Computer Security Center published a book called “The U.S. Department of Defense Trusted Computer System Evaluation Criteria”, better known as “The Orange Book” which defined seven levels of trust. The levels from lowest to highest are:

- **D** – Minimal protection
- **C1** – Some access control and permissions. Passwords required for logon
- **C2** – Authentication is audited and audit logs are held securely
- **B1** – Mandatory access control with labels. Security access is based on labels

- **B2** – Fully documented configuration control, facility management and system configuration. Security administration and operator functions are separated
- **B3** – Access control lists and full system documentation. Access is based on access control lists and labels
- **A** – Requires formal proof of the security of the system

Note that each subsequent level builds on the previous level.

To put the above into perspective, a normal PC is rated at *D* and Trusted Solaris at *B1*. A Solaris system running the Basic Security Module (BSM) is rated at *C2*.

In addition to these formal evaluations, the Common Criteria Organization has provided seven similar evaluations, called Evaluation Assurance Levels (EAL), which equate to the seven "Orange Book" levels. These levels are:

- **EAL1** – Functionally tested
- **EAL2** – Structurally tested
- **EAL3** – Methodically tested and checked
- **EAL4** – Methodically designed, tested and reviewed
- **EAL5** – Semi formally designed and tested
- **EAL6** – Semi formally verified design and tested
- **EAL7** – Formally verified design and tested

Further information on the common criterial EALs can be found at:

http://www.commoncriteriaportal.org/public/files/ccintroduction.pdf

Evaluation standards are of benefit because they provide formal validation of an operating system, allowing the purchaser to know whether a proposed solution will address the security requirements. The standards are also international and will not change from one country to the next.

Invalidating a Certification

There are a number of ways in which a certified operating system might be invalidated:

- By installing third party software that is not validated at the equivalent (or higher) security level.
- By installing operating system patches other than the recommended security patches. Sun provides a list of irrelevant patches that can safely be applied and will not affect certification. Any other patches might affect the certification.

➤ By operating lax permissions and revealing passwords

Detection and Device Management

This section looks at logging important system messages so that incidents can be recorded, including the use of the process accounting facility that comes with the standard Solaris 9 installation. Also, the Solaris Basic Security Module (BSM) is described here showing how to configure BSM and conduct an audit, as well as interpreting the results. The final part in this section looks at protecting access to devices.

Monitoring Login Attempts

Loginlog

By default, unsuccessful login attempts are only recorded after 5 attempts have been made. This is far too many because the system administrator should be alerted to the fact that incorrect passwords are being entered so that it can be carefully monitored. To log all unsuccessful login attempts, you need to first create the file where these messages will be logged, namely */var/adm/loginlog*, set the permissions to *700* and then modify two lines in the file */etc/default/login*.

```
# touch /var/adm/loginlog
# chmod 700 /var/adm/loginlog
```

This second command modifies the permissions so that only user *root* can access the file. Messages are logged to this file via *syslog*, described later in this section, in the *AUTH.NOTICE* category.

Change the lines:

SYSLOG_FAILED_LOGINS=5

RETRIES=5
To

SYSLOG_FAILED_LOGINS=0

RETRIES=1

Notice that by default the two lines are commented out. Remember to uncomment them as well. Also note that if you set the *RETRIES* variable to *0*, then you might not be able to login at all, except at the console.

It should be noted that login attempts using CDE (*dtlogin*) will not be caught by this facility. Only attempts that use the *login* command will be noticed.

lastlog, utmpx, wtmpx and last

The files */var/adm/utmpx* and */var/adm/wtmpx* record information about who is logged in to a system. *utmpx* contains current information and *wtmpx* contains historical information.

The file */var/adm/lastlog* records the prior login information. It is not an ASCII readable file. The example below shows the output received when user *john* logs in using *telnet*, the last time the user logged in is displayed on the screen:

```
SunOS 5.9

login: john
Password:
Last login: Fri Apr  9 01:40:27 from 192.168.1.2
```

Users should be made aware of this information as it could provide valuable information on unauthorized accesses to the system.

The *last* command (*/usr/bin/last*) displays login information from *utmpx* and *wtmpx*, including details of system reboots. The example below shows some truncated output from the *last* command:

```
# last
john      pts/1        test.mobileven Fri Apr  9 11:10 still logged in
root      console                     Fri Apr  9 01:51 - 01:52  (00:00)
john      dtremote       192.168.1.2: Fri Apr  9 01:40 - 01:42  (00:01)
john      dtremote       192.168.1.2: Mon Apr  5 22:47 - 01:12  (02:24)
reboot    system boot                 Sun Apr  4 16:41
```

Notice how the output indicates users that are “still logged in”.

System Log Files

There are three main log files used by the system to record important messages, *syslog*, which is described in the next subsection, */var/adm/messages* and */var/adm/sulog*.

/var/adm/messages

This file contains system messages and is the central repository for the majority of messages that would interest the system administrator. The type of messages that get logged here include:

- System boot messages

- Hardware error messages
- Failed *su* attempts
- User login failures
- System software and application error conditions
- Notification of *root* logins

Note that successful *su* attempts, when a user enters the correct password, are NOT recorded in this file.

SU logging

When a user uses the *su* command to become (substitute user), it is logged by default to the file */var/adm/sulog*. The behavior of this logging activity is controlled by the file */etc/default/su*. The *syslog* facility (described next) can also log *su* attempts via the AUTH facility, both successful and unsuccessful.

A sample */var/adm/sulog* appears below:

```
# cat /var/adm/sulog
SU 04/12 22:43 + pts/5 john-root
SU 04/12 22:46 + pts/5 john-testuser
SU 04/12 22:47 - pts/5 john-testuser
SU 04/12 22:48 - pts/5 john-root
SU 04/12 22:53 - pts/5 john-testuser
```

Note that a "+" in column 4 indicates a successful *su* and a "**-**" an unsuccessful attempt. It can be useful in situations where this file contains many entries, to search specifically for this minus sign to identify quickly any failed attempts.

Syslog

System logging is managed by the *syslog* facility. The daemon that runs is */usr/sbin/syslogd* and it is started by the startup script */etc/rc2.d/S74syslog*, a link to */etc/init.d/syslog*.

Messages are categorized by a facility and, within each facility, a priority. The facilities are:

- **kern** – Messages concerning the kernel
- **user** – Messages concerning user processes
- **mark** – Messages concerning timestamp information
- **auth** – Messages concerning authorization (*login*, *su* for example)

- **daemon** – Messages concerning daemon processes (*syslogd*, *inetd* for example)
- ***** - All of the facilities
- **local.*** – Locally defined message criteria

The priorities are:

- **emerg** – Emergency situations. These are broadcast to all users
- **alert** – Situations that need to be addressed immediately
- **crit** – Critical warnings
- **err** – Non-critical error messages
- **warn** – Warning messages
- **notice** – Other conditions that aren't errors, but might still require attention
- **info** – Messages for information only
- **debug** – Messages providing information when debugging processes
- **none** – Do not log any messages for this facility

You will often find that third party supplied software makes use of the locally defined facilities to provide *syslog* compatibility for their product.

Configuring Standalone syslog

By default, when you install Solaris 9, an entry for the system being installed is made in the */etc/inet/hosts* file. This also includes an alias *loghost*, which is provided for use with *syslog*. The *syslog* utility is configured through a configuration file, */etc/syslog.conf*, the standard file distributed with the Solaris 9 release is shown here:

```
# cat /etc/syslog.conf
#ident  "@(#)syslog.conf        1.5     98/12/14 SMI"   /* SunOS 5.0 */
#
# Copyright (c) 1991-1998 by Sun Microsystems, Inc.
# All rights reserved.
#
# syslog configuration file.
#
# This file is processed by m4 so be careful to quote (`') names
# that match m4 reserved words.  Also, within ifdef's, arguments
# containing commas must be quoted.
#
```

```
*.err;kern.notice;auth.notice                    /dev/sysmsg
*.err;kern.debug;daemon.notice;mail.crit         /var/adm/messages
*.alert;kern.err;daemon.err                      operator
*.alert                                          root
*.emerg                                          *

# if a non-loghost machine chooses to have authentication messages
# sent to the loghost machine, un-comment out the following line:
#auth.notice                ifdef(`LOGHOST', /var/log/authlog,
@loghost)

mail.debug                  ifdef(`LOGHOST', /var/log/syslog,
@loghost)

#
# non-loghost machines will use the following lines to cause "user"
# log messages to be logged locally.
#
ifdef(`LOGHOST', ,
user.err                                         /dev/sysmsg
user.err                                         /var/adm/messages
user.alert                                       `root, operator'
user.emerg                                       *
)
```

Note the following about the output above:

- Multiple facilities and priorities can be assigned to a single entry
- The action column on the right hand side can be to write to a file, a device, or to send email to specified users
- Conditions can also be applied to entries, for example, only if LOGHOST is defined (a *loghost* entry is present in the */etc/inet/hosts* file)
- The last six lines define actions to take if LOGHOST is not defined, so that messages are still written locally if this situation is encountered
- By default, the *auth.notice* entry is commented out. It is a good idea to log all authorization messages to the file */var/log/authlog*, because it makes it easier to spot important login failure messages

Configuring syslog to Log Centrally

A professional attacker will try to cover his/her tracks by modifying the system logs so that there is no evidence that an attack even took place. This is done quite easily if the attacker has gained privileged

access to your system. However, if you configure *syslog* to send its messages to one or more central logging servers, then this is made infinitely more difficult, if not impossible, thereby preserving your evidence.

TIP: On any central logging servers, disable ALL services, except *syslog* on UDP port 514. This prohibits ANY access to the servers other than the logging messages. Also, configure more than one logging server to avoid having a single point of failure.

To log centrally, you need to do two things:

1. For each logging server, add an entry in */etc/inet/hosts* for the server and also append the alias *loghost* to each one

2. Add extra entries for each category/priority to be logged to these servers with the action column being *@hostname*, where *hostname* is the name of a central logging server

So, taking the first two entries from the sample */etc/syslog.conf*, and to centrally log these two entries to servers named *bill* and *ben*, the resulting configuration entries would look like this:

```
*.err;kern.notice;auth.notice                /dev/sysmsg
*.err;kern.notice;auth.notice                @bill
*.err;kern.notice;auth.notice                @ben
*.err;kern.debug;daemon.notice;mail.crit     /var/adm/messages
*.err;kern.debug;daemon.notice;mail.crit     @bill
*.err;kern.debug;daemon.notice;mail.crit     @ben
```

Using Multiple Files

Instead of having most of the messages being written to */var/adm/messages*, you can specify different files to log different messages. This does make the configuration slightly more complex, but it should be easier to manage and easier to find specific messages. To log all authorization messages, for example, to a file named */var/log/authlog*, you could add the following entry to */etc/syslog.conf*:

```
auth.*                                       /var/log/authlog
```

Note that TABS must be used to separate the fields.

Remember to make the *syslog* daemon re-read its configuration file after making changes by executing:

```
# pkill –HUP syslogd
```

Process Accounting

Process accounting is installed as part of a default Solaris 9 installation and, although it is primarily designed as an accounting tool for billing uses, it also has value as a security monitoring tool. The process accounting package helps with the following:

- Assisiting with the overall security of a system because of the logging facility it provides, including start and end times of a command being executed as well as the command name and the terminal name from which it was run
- Monitoring the usage of the system in terms of processor, memory and disk usage
- Monitoring for performance issues and capacity planning
- Troubleshooting a number of system problems, some of which could be as a result of an attack taking place
- Providing additional evidence of when a user was logged in and logged out

Process accounting is a good and useful package, but the following facts should be considered about this package:

- Process accounting is a historical view of what happened, it is not a real-time audit of what's going on now
- Accounting records are only written once a command being run has completed. For long running programs, like a password cracker for example, an entry won't appear in the accounting files until it's finished
- Accounting contains the name of the program being run, but the program is not validated. If a spoofed version of the *login* program was being used for example, this would not be noticed
- Accounting records can only be used as part of an investigation after an attack has taken place

Auditing with the Basic Security Module (BSM)

This section looks at auditing the Solaris environment. It describes two main functions, namely recording events that occur and also managing the allocation and security of devices.

Overview

The daemon process that runs is */usr/sbin/auditd* and the configuration files can be found in the */etc/security* directory. The following configuration files are used in the auditing process:

- */etc/security/audit_startup* – Sets initial policy for the process
- */etc/security/audit_control* – Controls the type of action to be audited and includes such items as where the data files are stored and the minimum amount of free disk space that must exist to allow auditing to continue

- */etc/security/audit_user* – Provides more detailed control allowing specific users and actions to be audited
- */etc/security/audit_event* – Defines the events that can occur
- */etc/security/audit_class* – Groups events into classes for easier management
- */etc/security/audit_data* – Contains the current *pid* for the auditing daemon and the full pathname for the current audit log file

The file */etc/security/audit_startup* is read when the daemon process is started and sets general policy values. One such value is:

```
auditconfig -setpolicy +cnt
```

which instructs the audit daemon to drop records if resources are exhausted (such as running out of disk space). This is preferably to processes being suspended instead.

Here's an example of a single class entry in */etc/security/audit_class*

```
0x00001000:lo:login or logout
```

and below is the relevant contents of the */etc/security/audit_event* file that relates to the *lo* class specified above. You can see how the grouping of events into a class makes it easier to audit specific types of information:

```
6152:AUE_login:login - local:lo
6153:AUE_logout:logout:lo
6154:AUE_telnet:login - telnet:lo
6155:AUE_rlogin:login - rlogin:lo
6158:AUE_rshd:rsh access:lo
6159:AUE_su:su:lo
6162:AUE_rexecd:rexecd:lo
6163:AUE_passwd:passwd:lo
6164:AUE_rexd:rexd:lo
6165:AUE_ftpd:ftp access:lo
6171:AUE_ftpd_logout:ftp logout:lo
6172:AUE_ssh:login - ssh:lo
6173:AUE_role_login:role login:lo
6212:AUE_newgrp_login:newgrp login:lo
6213:AUE_admin_authenticate:admin login:lo
```

Enabling BSM

There are three steps in enabling the auditing facility:

- Run the utility */etc/security/bsmconv*
- Edit the */etc/security/audit_startup* file, if required (this file is only created when you run *bsmconv*)

- Reboot the system to bring it up with auditing enabled

```
# /etc/security/bsmconv
This script is used to enable the Basic Security Module (BSM).
Shall we continue with the conversion now? [y/n] y
bsmconv: INFO: checking startup file.
bsmconv: INFO: move aside /etc/rc3.d/S81volmgt.
bsmconv: INFO: turning on audit module.
bsmconv: INFO: initializing device allocation files.

The Basic Security Module is ready.
If there were any errors, please fix them now.
Configure BSM by editing files located in /etc/security.
Reboot this system now to come up with BSM enabled.
```

The following files are created in the */etc/security* directory when you enable BSM:

- audit_startup
- device_allocate
- device_maps

Note that the volume management facility conflicts with BSM if you're going to be using it for securing devices (described later in this section) and is automatically disabled when BSM is enabled.

Generating an Audit

Suppose you want to audit all file deletions to catch someone maliciously deleting important files.

You have two choices how to do this:

- Edit the */etc/security/audit_control* file to audit for all users, i.e. non-attributable to a single user
- Edit the */etc/security/audit_user* file to audit for a specific user

In this example, I edited */etc/security/audit_user* to look specifically for user *root* deleting files. I added the *fd* option for this user and saved the file, as shown here:

```
root:fd,lo:no
```

Use the *audit* command to make the daemon, *auditd* re-read the configuration files. This command is described at the end of this section.

TIP: There is an *all* option to auditing, but this is not recommended for any period of time as it consumes vast amounts of disk space. If this option is to be used, then only leave it on for a few minutes to see how much data is gathered.

Interpreting the Results

Continuing the example scenario, you now want to inspect the audit file(s) to see if any files have been deleted by the *root* user. Use the *auditreduce* command to select only the records of interest and then pipe the result to *praudit* to present the data in readable form. The following command achieves this:

```
# auditreduce –a 20040412 –u root –c fd | praudit
header,127,2,unlink(2),,Tue Apr 13 10:27:34 BST 2004, + 917 msec
path,/etc/inet/hosts attribute,100444,root,other,32,2619,0
subject,root,root,other,root,other,521,390,0 0 ultra
return,success,0
```

The above command searches the audit data files (in */var/audit*) for the date *20040412*, the user *root* and the class *fd* (file deletions). It returns the item of interest showing that user *root* deleted the file */etc/inet/hosts* and that the deletion was successful.

There are other search criteria for use with *auditreduce*. Consult the man pages for detailed options.

The audit command

This command sends instructions to the *auditd* daemon process. It has three options:

- **-n** – Close the current log file and open a new one
- **-s** - Re-read the configuration files
- **-t** – Close the current log file and terminate

Audit Log Files

The default location for audit log files is */var/audit*. This is specified in the configuration file */etc/security/audit_control*. Each log file has the following format:

<startdatetime>.<enddatetime>.<hostname>

So, for a file that started at 0813 on 9 Apr 2004 and ended at 0906 on the same day for hostname "ultra", the file would be:

```
20040409081343.20040409090640.ultra
```

If the file is still being written to, i.e. the current log file, then it's name would take the form:

<startdatetime>.not_terminated.<hostname>

If the filename above was still the active file, then it's name would have been:

```
20040409081343.not_terminated.ultra
```

Note: A reboot of the system automatically causes the current log file to close and a new one to be opened when the system comes back up.

Disabling BSM

If you no longer want to run the auditing facility it can be easily disabled by running:

 # /etc/security/bsmunconv

```
bsmunconv: ERROR: this script should be run at run level 1.
Are you sure you want to continue? [y/n] y
This script is used to disable the Basic Security Module (BSM).
Shall we continue the reversion to a non-BSM system now? [y/n] y
bsmunconv: INFO: moving aside /etc/security/audit_startup.
bsmunconv: INFO: restore /etc/rc3.d/S81volmgt.
bsmunconv: INFO: removing c2audit:audit_load from /etc/system.
bsmunconv: INFO: stopping the cron daemon.

The Basic Security Module has been disabled.
Reboot this system now to come up without BSM.
```

Notice that the script should be run at single user or run level 1 because it makes system changes. It also restores the volume management facility. The system needs to be rebooted to complete the operation.

Device Management

BSM contains a feature that protects devices attached to the system. It does the following:

- Stops multiple users from accessing a device simultaneously
- Stops anyone else from reading the data you might have just written to a device
- Stops anyone else from overwriting your data on a device
- Prohibits anyone else from getting information or data from a device after you have used it

Device management maintains some files to achieve this:

- **/etc/security/device_allocate** – A file that contains access control information about each device
- **/etc/security/device_maps** – Associates physical devices with logical file names
- **/etc/security/dev** – A directory containing all the relevant device files and used for locking

There are also some commands associated with BSM device management:

- **allocate** – Used to allocate a specific device to a user

- **deallocate** – Used to deallocate a device after a user has finished with it
- **dminfo** – Used to report information on a device. Reads the *device_maps* file
- **list_devices** – Produces a list of allocatable devices
- **device-clean scripts** – A series of scripts that prohibit any other user from accessing information or data from a device when the user has finished using it. The scripts can be found in the directory */etc/security/lib*.

Authorizing Device Access to a User

In order for a user to be able to user an allocatable device, certain authorizations must be given to the user. These are done using the *usermod* command. The authorizations are already present in the */etc/security/auth_attr* file.

To give user *testuser* the device authorizations, run the following command:

```
# usermod –A "solaris.device.*" testuser
```

This, in turn, makes an entry in the user attributes database, */etc/user_attr* as shown here:

```
testuser::::type=normal;auths=solaris.device.*
```

Note that the *auth_attr* and *user_attr* are both databases that are used as part of the Role Based Access Control (RBAC) feature, described later in this document.

Managing Devices Under BSM Control

There are a number of steps to follow to restrict access to specified devices. These are described below:

1. Ensure any devices have an entry in the file */etc/security/device_maps*
2. Edit the file */etc/security/device_allocate* to determine which devices can be allocated by users
3. Any user that is going to try and use these devices must have the necessary authorizations. Only users with these authorizations can use the devices, all others will not be able to allocate them
4. An empty lock file needs to be created for each of the devices you decide can be allocated by a user. The lock file must be created in the */etc/security/dev* directory and should be the same name as that used when adding the device to the */etc/security/device_maps* file
5. Change the permissions of the actual device files (in */dev*) to 000 and make sure they are owned by user *bin* and group *bin*

Security Attacks

This section looks at different types of attacks that can be attempted against your systems or network. It also looks at ways in which these can be detected and prevented.

Denial of Service (DoS) Attacks

A DoS attack is one where the resources of a system (or network) become depleted so as to prevent the normal operation of that system (or network). As the name implies it denies service to legitimate users of the system. A DoS attack can be mailicious or accidental and normally involves using up all the file space, network bandwidth, swap space, memory, processor cycles or the number of processes that can run on the system. Some of the more popular DoS attacks are described below:

- **Worm** - A worm is a deliberate attack on a system where a program replicates itself over and over again, either on the same system, or between systems, thereby spreading the attack. This type of attack will often take over a system and use all of its processor resources to continue spreading the worm
- **Fork bomb** – These processes keep replicating themselves (spawning new processes) until the system reaches its limit for the number of processes that can run. At this point the system will not be able to create any new processes, stopping users from doing anything. This kind of attack is normally malicious, but could also be accidental if, for example, a programmer writes some code using recursion that is not quite right. In this instance, a legitimate program could have exactly the same effect
- **Ping of death** – This causes a system to crash when a *ping* request is received containing a larger amount of data than is permitted, normally over 64K
- **TCP SYN** – This attack exploits the TCP three-way handshake by leaving half-open connections. It does this until the target system is unable to open any more connections
- **Teardrop** – This exploits the TCP fragmentation of packets facility by sending invalid offset values in fragmented packets. The receiving system hangs when trying to reassemble the packets
- **Smurf** – This attack sends a broadcast *ping* to all hosts on a network, but substitutes the target system's address for replies to be sent, thereby overloading the target system
- **Filling up system logs** – This can be an accidental DoS if a user, or programmer, does something that causes the system to repeatedly log an error. If the system does not have a separate */var* filesystem, it can hang the entire system
- **Backing up to a file** – A small typing error can result in a backup writing to a file instead of a backup device. The backup files can be extremely large and can quickly consume vast amounts of disk space

Preventing DoS Attacks

Some DoS attacks can be prevented fairly easily, whilst for others there is little protection. The following points can be used to assist with stopping some of the attacks listed in the previous section:

- TCP SYN and Ping of Death attacks use ICMP messages. If you have a firewall installed, then restrict, or disable the use of ICMP through the use of the firewall rules. You can also run an Intrusion Detection System, such as Courtney to detect unusual amounts of activity
- Smurf attacks rely on replies being sent to broadcast ICMP messages. You can either disable the use of ICMP on your firewall, or specifically stop your system from responding to broadcast requests. Do this by setting the following tunable parameters:

```
# ndd -set /dev/ip ip_respond_to_echo_broadcast 0
# ndd -set /dev/ip ip_forward_directed_broadcasts 0
```

- Fork bombs can have their effect reduced by setting the maximum number of user processes to a specified value. This would be done by setting *maxuprc* to say *75* in */etc/system* and would not allow a user to have more than the specified number of processes. It would prevent any single user from being able to use up all the available processes on a system
- You can stop a user from being able to use up all of the disk space by installing quotas on relevant filesystems
- Use of the *ulimit* command can stop users from being able to hog system resources
- For the example of backing up to a file instead of the intended backup device, put the backup procedure into a script to remove human error
- Monitor disk space regularly, or better still, install a network management system, such as HP OpenView, or BMC Patrol so that you get early warning of filesystems filling up.
- For system log files, ensure that you have a regular log rotation strategy that is suitable for your site and the amount of information being logged. From Solaris 9, the *logadm* utility exists for this purpose

How DoS Attacks Execute

Network DoS attacks can be executed remotely from another system, or host out on the Internet. It involves the attacker merely knowing the address to send the attack to, which in most cases, can be found out very easily. The attacker then modifies the packet to be sent, in the case of a smurf attack, making the target system the address to reply to. For attacks like *ping of death*, it is the action of sending loads of packets with much larger than expected amounts of data that causes an attack to be successful.

Note that Solaris Sparc systems are not vulnerable to *ping of death*, but Solaris x86 systems are.

Most host-based DoS attacks will involve a malicious piece of code being installed on the target system, which means that the attacker must have gained access to the system in order to install the program. Once installed, the program can be triggered to execute either when a certain condition is encountered, or for example, after a specified time has elapsed.

Privilege Escalation Attacks

Types of Attack

- **Trojan Horse** – As the name implies, this exploit involves installing, or modifying a legitimate program to perform not only its real actions, but some additional ones too. It is these additional actions which undermine the security of the system and allow unauthorized access. The attacker must have already gained access (or have legitimate access) to be able to install a *Trojan Horse*
- **Buffer Overflow** – This exploit is done through programming where, for example, you store 30 characters in a buffer only defined to take 15. In this case, the stack entries become corrupt allowing the programmer to introduce new code to be executed at a different return location. The return code could be a Unix shell with *root* privileges for example. Some Solaris services suffer from buffer overflow vulnerabilities, which might be exploited by an attacker, but the majority can be prevented if current patches are installed on the system. If an attacker has already gained access to the system, and has access to a compiler, then it is potentially very easy for the attacker to gain privileged access
- **Backdoor** – This provides an alternative entry point to a system that is not publicly known, whether it is malicious or accidental. Programmers often leave *backdoors* in their code to allow additional debugging, and sometimes forget to take it out when the program becomes operational. These could simply be the creation of an extra *root* user through to changing ownership and permission of say a physical device file to gain access. An accidental *backdoor* could be exploited by an intuitive attacker, but normally access has to be obtained first in order to be able to install a *backdoor*
- **Rootkit** – A rootkit is an entire package for not only accessing a system, but covering the tracks once inside. An attacker doesn't need to be a system expert to use one of these, someone else will have done all the hard work, the attacker just uses it (rather like a script kiddie). A rootkit will typically contain utilities to remove log entries for example. Centralized logging negates a lot of the rootkit functionality. Using a rootkit successfully allows an attacker to remain unnoticed and then to leave a *backdoor* for future attacks. It should be noted that the attacker must already have gained access to the target system in order to be able to install the *rootkit*
- **Loadable Kernel Module** – A different kind of *rootkit* where the live running kernel is exploited. The attacker must first have gained *root* access and then installs two kernel modules. One of these installs the utilities needed for the attack and the second makes sure that the loaded modules do not appear on a *modinfo* listing, if run by the system administrator
- **Symbolic Links** – Symbolic links (or soft links) are extremely useful, but can also be very dangerous. A system administrator could unintentionally leave a symbolic link to a *root* owned file for example, allowing an attacker instant access to privileged data or programs. An attacker must already have gained access to the system for this exploit to be used

Detecting Attacks

There are various methods for detecting that an attack has taken place. This section looks at detecting *backdoor* and *Trojan Horse* attacks.

Using Solaris Fingerprint Database

The fingerprint database supplied by Sun Microsystems provides the facility to check that Solaris Operating Environment files have not been tampered with, or modified by an unauthorized intruder. For single files, you can use the interactive option on Sun's web site at:

http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl

First though, you need the MD5 binary to create a local MD5 signature that can be checked against the one held by Sun Microsystems. Get this from:

http://sunsolve.sun.com/md5/md5.tar.Z

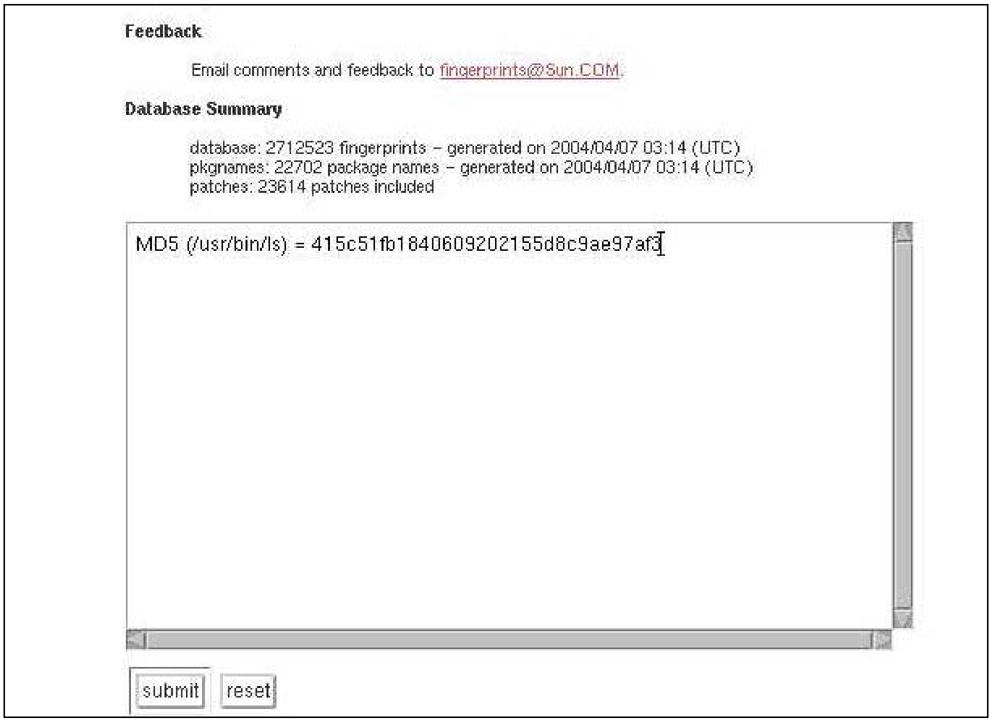
For this example, I ran

```
# md5-sparc /usr/bin/ls
```

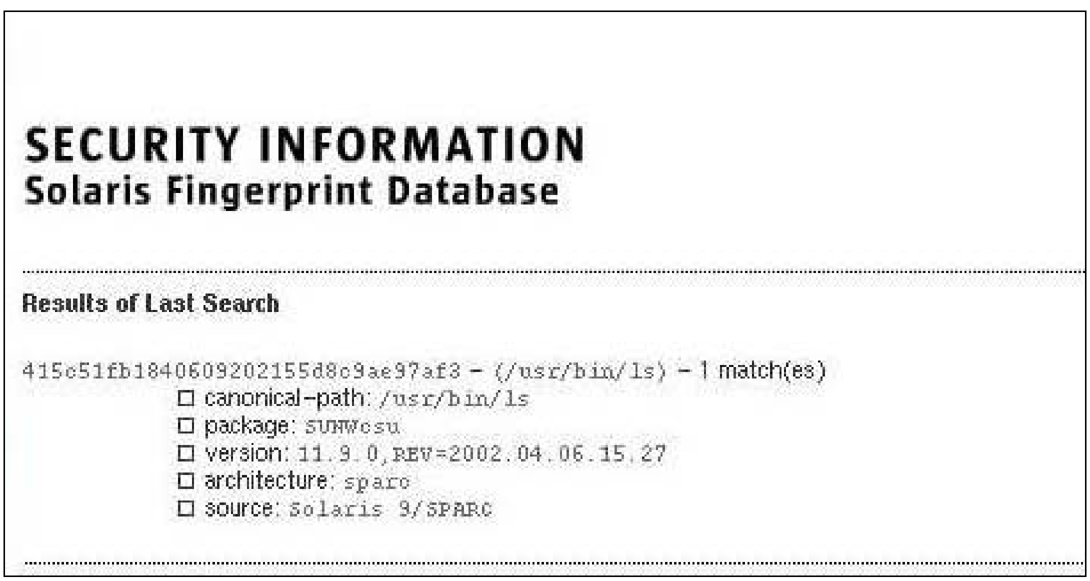
to obtain the md5 signature for the *ls* command.

Then, startup the interactive fingerprint script and paste in the result from the previous command.

The partial screenshot below shows the relevant section of the screen:



The result is shown in the next screenshot.



Note that the checksums match and the *1 match(es)* indicates this too.

The interactive method is quite labor intensive, if you want to check a larger number of files, so you can download the Solaris Fingerprint companion and the *sidekick* utility from

http://wwws.sun.com/software/security/downloads.html

This method allows a number of MD5 signatures to be generated and automatically fed to the Solaris Fingerprint database for comparison. If any of the files being checked do not match the Sun Microsystems version, then it has been modified or tampered with, and needs to be investigated.

Using File Listings

This is a very simple way of detecting whether additional, unexpected files have been installed on a system. It involves storing a listing of specified directories, such as */usr/bin* for example. It is not a serious deterrent to an attacker because the file list itself could be modified by an attacker if it is left on the same system. This method would not always detect modifications to files or programs, but would notice if new files have been installed. It works by taking a listing of a directory, saving it to a text file, then taking the same listing at some specified interval and running *diff* against the two files to highlight any differences.

Using Checksums

Checksums constitute a more secure method of detecting *rootkits* or *backdoors*, but is still relatively easy for an attacker to overcome. This method works by listing the files in a file system and creating a checksum for each file. The list needs to be written to a file to be compared with a future run to highlight any differences. The main reason for this method being insecure is that many *rootkits* available to attackers, also contain methods of installing *Trojan* files, whilst maintaining the same access times and checksums, thereby preventing their detection by this means. The following example shows the checksum output produced by the file */usr/bin/su*:

```
# sum /usr/bin/su
2341 44 /usr/bin/su
```

Using File Digests

Using a file digest mechanism is much more secure and offers a much higher degree of confidence of knowing whether your file system has been compromised. MD5 is a good example and works by creating a unique 'fingerprint' of a file. The following example shows the md5 signature created from the file */usr/bin/su*:

```
# ./md5-sparc /usr/bin/su
MD5 (/usr/bin/su) = 2304d7ee650512ed3c61f74a2ceb76f1
```

Using the *find* Command

If you do not have access to a fingerprinting tool, then the *find* command is the next best utility for detecting unauthorized access to a system. The *find* command has several useful options, the relevant ones for this scenario include:

- Checking files owned by the *root* user
- Checking for set-uid and set-gid programs
- Looking for recently modified files

The following examples show how to use each of these facilities:

- The first example demonstrates how to look for files owned by the *root* user in a user's home directory – somewhere you wouldn't expect to find them!

```
# find /export/home/testuser -user root -exec ls -l {} \;
-rw-r--r--  1 root  staff  19084 Apr 8 23:35 /export/home/testuser/ls
```

The listing has revealed a copy of the *ls* command residing in the home directory of the *testuser* user. Whilst these occurrences might prove to be quite innocent, or accidental, this command highlights items that should be investigated.

- This example shows all of the files in */usr/bin* that have the set-uid bit set and executes a long listing. For this check, you would be looking for unexpected additional set-uid programs that an attacker might install for later use. The example here shows a listing of a single directory, but you would normally want to widen the search to include complete file systems.

```
# find /usr/bin -perm -4000 -exec ls -l {} \;
-r-sr-xr-x   1 root   sys   12548 Feb 25  2003 /usr/bin/sparcv7/newtask
-r-sr-xr-x   2 root   bin   11248 Apr  7  2002 /usr/bin/sparcv7/uptime
-r-sr-xr-x   2 root   bin   11248 Apr  7  2002 /usr/bin/sparcv7/w
-rwsr-xr-x   1 root   sys   37824 Dec 13  2002 /usr/bin/at
-rwsr-xr-x   1 root   sys   13916 Apr  6  2002 /usr/bin/atq
-rwsr-xr-x   1 root   sys   12836 Apr  6  2002 /usr/bin/atrm
-r-sr-xr-x   1 root   bin   17180 Aug  7  2003 /usr/bin/crontab
-r-sr-xr-x   1 root   bin   14276 Apr  7  2002 /usr/bin/eject
-r-sr-xr-x   1 root   bin   25964 Apr  6  2002 /usr/bin/fdformat
-r-sr-xr-x   1 root   bin   29492 Apr  6  2002 /usr/bin/login
-rwsr-xr-x   1 root   sys    7616 Apr  6  2002 /usr/bin/newgrp
-r-sr-sr-x   1 root   sys   21964 Apr  7  2002 /usr/bin/passwd
-r-sr-xr-x   1 root   bin    9644 Apr  7  2002 /usr/bin/pfexec
-r-sr-xr-x   1 root   sys   22292 Apr  7  2002 /usr/bin/su
-r-s--x--x   1 uucp   bin   54740 Apr  7  2002 /usr/bin/tip
-r-sr-xr-x   1 root   sys   17432 Feb 25  2003 /usr/bin/sparcv9/newtask
```

```
-r-sr-xr-x   2 root   bin    15296 Apr  7  2002 /usr/bin/sparcv9/uptime
-r-sr-xr-x   2 root   bin    15296 Apr  7  2002 /usr/bin/sparcv9/w
-r-s--x--x   1 root   sys 1163504 Jul 30  2003 /usr/bin/admintool
-r-sr-xr-x   1 root   bin    21448 Feb 26  2003 /usr/bin/rcp
-r-sr-xr-x   1 root   bin    55292 Apr  6  2002 /usr/bin/rdist
-r-sr-xr-x   1 root   bin    15284 Apr  6  2002 /usr/bin/rlogin
-r-sr-xr-x   1 root   bin     9176 Apr  6  2002 /usr/bin/rsh
-r-s--x--x   1 root   lp      9872 Sep 19  2003 /usr/bin/cancel
-r-s--x--x   1 root   lp     22972 Apr  7  2002 /usr/bin/lp
-r-s--x--x   1 root   lp      9688 Apr  7  2002 /usr/bin/lpset
-r-s--x--x   1 root   lp     22820 Apr  7  2002 /usr/bin/lpstat
-r-sr-xr-x   1 root   sys    41416 Apr  6  2002 /usr/bin/chkey
-r-sr-xr-x   1 root   bin     4832 Apr  7  2002 /usr/bin/mailq
-r-sr-xr-x   1 root   bin    38732 Apr  7  2002 /usr/bin/rmformat
-r-sr-xr-x   1 root   bin     6204 Apr  7  2002 /usr/bin/volcheck
-r-sr-xr-x   1 root   bin    12620 Apr  7  2002 /usr/bin/volrmmount
-r-sr-xr-x   1 root   bin   236492 Mar 14  2003 /usr/bin/pppd
---s--x--x   1 root   uucp   69552 Apr  6  2002 /usr/bin/ct
---s--x--x   1 uucp   uucp   83340 Apr  6  2002 /usr/bin/cu
---s--x--x   1 uucp   uucp   66788 Aug  8  2003 /usr/bin/uucp
---s--x--x   1 uucp   uucp   22676 Apr  6  2002 /usr/bin/uuglist
---s--x--x   1 uucp   uucp   19576 Apr  6  2002 /usr/bin/uuname
---s--x--x   1 uucp   uucp   61184 Aug  8  2003 /usr/bin/uustat
---s--x--x   1 uucp   uucp   70908 Aug  8  2003 /usr/bin/uux
-rwsr-xr-x   1 root   bin    52976 Apr  6  2002 /usr/bin/cdrw
```

- The last example identifies files in the */etc* directory owned by the *root* user that have been modified in the last two days:

 # find /etc -user root -mtime -2 -print

```
/etc/passwd
/etc/inet/inetd.conf
```

As you can see from this example, there has been a modification to the */etc/inet/inetd.conf* file, which could indicate an unauthorized service might have been enabled as a *backdoor*. Also note that the *passwd* file has been modified, maybe to create a new account. Either way, it would prompt the system administrator to investigate further.

Using *Tripwire*

Tripwire is a third party product available from www.tripwire.com that produces a fingerprint of specified directories in a system and creates a database that can be checked to report changes to files. Tripwire does not just report on modifications to files, it also notices changes to the attributes of files, such as access time and modification time.

The following output shows the result of a Tripwire report after running a check on the fingerprint database. Before running the check, the following commands were run to force two changes:

- **touch /etc/passwd** – to update the access time on the password file
- **cp /etc/inet/hosts /etc/inet/hosts.JP** – to make a copy of the *hosts* file

```
# bin/twprint --print-report --report-file ultra-20040413-165329.twr

Note: Report is not encrypted.
Tripwire Integrity Check Report version 4.0.0
Tripwire(R) for Servers version 4.1.0.210

Report generated by:          root
Report created on:            Tue, 13 Apr 2004 16:53:29 +0100
Database last updated on:     Tue, 13 Apr 2004 16:46:10 +0100
====================================================
Report Summary:
====================================================
Host name:                    ultra
Host IP address:              192.168.1.2
Host ID:                      0x80888ae4
Policy file used:             /usr/local/tripwire/tfs/policy/tw.pol
Configuration file used:      /usr/local/tripwire/tfs/bin/tw.cfg
Database file used:           /usr/local/tripwire/tfs/db/database.twd
Command line used:            ./tripwire --check
====================================================
Rule Summary:
====================================================
-----------------------------------------------------------------------
  Section: Unix File System
-----------------------------------------------------------------------
  Rule Name               Severity Level    Added    Removed  Modified
  ---------               --------------    -----    -------  --------
  Tripwire Data Files           100         0          0        0
  Other Filesystems             100         0          0        0
  System Devices                100         0          0        0
  User Home Directories         35          0          0        0
  Mounted Filesystems           100         0          0        0
  (/mnt)
  System Processes              100         0          0        0
  (/proc)
  Temporary directory           35          0          0        0
  (/tmp)
  Variable System Files         35          0          0        0
  Tripwire Binaries             100         0          0        0
  System Binaries               100         0          0        0
  Library Files                 35          0          0        0
```

```
  Include Files                35          0            0         0
  Man Pages                    35          0            0         0
  Administrative Binaries      100         0            0         0
* System configuration files   100         1            0         1
  System Directories           100         0            0         0

Total objects scanned:  28489
Total violations found:  2

====================================================
Object Detail:
====================================================
-----------------------------------------------------------------------
  Section: Unix File System
-----------------------------------------------------------------------
-----------------------------------------------------------------------
Rule Name: System configuration files (/etc)
Severity Level: 100
-----------------------------------------------------------------------
  ----------------------------------------
  Added Objects: 1
  ----------------------------------------
Added object name:  /etc/inet/hosts.JP
  ----------------------------------------
  Modified Objects: 1
  ----------------------------------------
Modified object name:  /etc/passwd
      Modify Time Expected  Tue, 13 Apr 2004 11:06:04 +0100
*                 Observed  Tue, 13 Apr 2004 16:53:25 +0100
      Change Time Expected  Tue, 13 Apr 2004 11:06:04 +0100
*                 Observed  Tue, 13 Apr 2004 16:53:25 +0100
====================================================
Error Report:
====================================================
No Errors
-----------------------------------------------------------------------
*** End of report ***

Report generated by:
Tripwire(R) for Servers version 4.1.0.210 for Solaris (SPARC) Operating
Systems
Tripwire is a registered trademark of Tripwire, Inc. All rights
reserved.
```

Note that the report displayed information on both of the forced changes made immediately prior to running the check.

Kernel Trust and OpenBoot

The kernel is implicitly trusted because it IS the operating system. For this reason, the kernel is potentially vulnerable to attacks, because, once compromised, an attacker has full control of the system.

At system boot time, kernel modules are loaded from these directories:

- **/platform/`uname –i`/kernel/**
- **/platform/`uname –m`/kernel/**
- **/kernel**
- **/usr/kernel**

To protect the kernel as much as possible, these directories should be checked regularly and audited to make sure there are no unauthorized additions.

You should also check */etc/system* because this file is used to load modules as well. Make sure this file is properly protected and inspected regularly. Using a product such as *Tripwire* will monitor any changes to the file.

OpenBoot

The OpenBoot PROM is the low level system interpreter that is often unprotected. If an attacker gains access to the system console, then this might be your only defense, but only if it is properly secured.

By default, the OpenBoot PROM comes completely unsecured, which means a command can be issued to boot from a different kernel file, boot across a network, the potential for compromise is endless.

There are two settings that need to be addressed to secure the OpenBoot console so that anyone gaining access to the console can reboot the system, but cannot alter any configuration parameters. The first is *security-mode*:

There are three levels of OpenBoot EEPROM security:

- None – There are no restrictions and any command can be used without entering a password. This is the default state
- Command – Restricted access with the user only being able to enter the *boot* or *continue* commands without a password. All other commands require a password
- Full – The highest security level where the user can only enter the *continue* command without a password

To set the security to the highest level, enter the following as *root* on the running system:

eeprom security-mode=full

The second setting is the EEPROM password, set this by executing

eeprom security-password=

This prompts the user to enter a password twice.

Note: Setting the EEPROM password should not be done lightly because it cannot be reset easily if forgotten and could render the system useless. The EEPROM device would have to be removed and reprogrammed – this must be done by Sun Microsystems.

File and System Resources Protection

This section is concerned with user accounts and how to protect them from intruders, as well as restricting access to files and the *root* account. It also describes Role Based Access Control (RBAC) allowing privileged functions to be carried out by regular users, without having to reveal the *root* password. Also, in this section, there is a brief discussion of Pluggable Authentication Modules (PAM) and Kerberos.

User Account Protection

User accounts and passwords are probably the most vulnerable to an attacker, so they should be guarded closely to make sure you are not leaving the front door open to your systems. This section looks at the ways in which user accounts can be better protected.

Checking for Duplicate Accounts

One tactic of the attacker is to create a user account with the same UID as an existing account, sometimes to make a clone of the *root* account. As an example, I have created a dummy account with the username *roothack* and a UID of 0, which gives this user the same privileges as the *root* user. Use the *logins* command to detect duplicate accounts as shown here:

```
# logins -d

root            0       other           1       Super-User
roothack        0       other           1
```

A duplicate user account cannot be created using the *useradd* program, because the UID is already in use, it has to be created manually.

The only scenario where a duplicate account can possibly be considered is where more than one user needs to do the same thing and would otherwise have to share a single user account – but that would breach most, if not all, security policies. By far a better solution would be to use Role Based Access Control (RBAC) to create a role and then assign multiple users to the role. This would maintain consistency and still retain accountability, providing the ability to audit and log the actions carried out by each user.

Expiring Accounts

User accounts can be expired in three ways:

- After a specified time of inactivity
- On a specific date
- Immediately

You can also use a combination in that an account could be set to expire if it is not used for a specific number of days, but also expire on a certain date.

To expire the account *temptest* if there has been no activity for 2 days:

 # usermod –f 2 temptest

To expire the account *temptest* on April 20 2004:

 # usermod –e 04/20/2004 temptest

To expire the account *temptest* immediately, lock the passwd:

 # passwd –l temptest

Note that there is no immediate expiry option, you have to lock the account.

You can display the expiry information for a user account by running the *logins* program as shown here:

 # logins –l temptest –a

```
temptest        8888    staff           10      Temporary test User
                        -1 000000
```

The fields are explained as follows:

- **temptest** – The user name
- **8888** – The UID
- **staff** – The primary group
- **10** – The primary group ID (GID)
- **Temporary test User** – The user comment from */etc/passwd*
- **-1** – The inactivity flag (-1 means that this flag is not set)
- **000000** – The expiry date (all zeros means the account will not expire)

If you run this again after setting the inactivity flag to 2 days and the account to expire on 20 April 2004, the values will change:

 # logins –l temptest –a

```
temptest        8888    staff           10      Temporary test User
                        2 042004
```

To set a user account so that it will no longer expire on the specified date, use a null string “”, and to turn off the inactivity flag, set it to 0.

Note: The expiry information is stored in */etc/shadow*.

Restricting *root* Logins

It is bad practice to allow *root* to login directly across the network. The only time that *root* should be able to login, is at the console.

To prevent remote logins as *root*, make sure the following entry in */etc/default/login* is not commented out (does not have a "#" in column 1):

```
CONSOLE=/dev/console
```

The only way for a user to become *root* is to use the *su* command after having first logged in as a normal user. This makes it more difficult for an attacker, because there are two passwords that have to be compromised in order to gain *root* access.

Dormant Accounts

When someone leaves the organization, their account should be deleted immediately, or at least locked, and the files either archived, or moved to another user, or deleted.

To lock the account *temptest*:

```
# passwd –l temptest
```

To delete the account *temptest*:

```
# userdel temptest
```

To delete the account *temptest* and remove the home directory (including files:

```
# userdel –r temptest
```

You should also look for any other files that the user may have owned that did not reside in the home directory, because these files will become a security risk if the user account is deleted. You can use the *find* command to change the ownership of these files. In the example below, the files owned by user *temptest* are changed to be owned by *root*:

```
# find / -user temptest -print

find: cannot find temptest name
```

Notice that *find* cannot identify the user *temptest* because it has already been deleted. In this instance, use the UID, which was *8888*, instead:

```
# find / -user 8888 -print -exec chown root {} \;
/var/report1
/var/report2
/var/report3
```

Now list the files to check they have changed owner:

```
# ls -l /var/report*
-rw-r--r--   1 root     other          0 Apr 10 22:34 /var/report1
-rw-r--r--   1 root     other          0 Apr 10 22:34 /var/report2
-rw-r--r--   1 root     other          0 Apr 10 22:34 /var/report3
```

Protecting Passwords

The security policy should provide users with guidelines for passwords, including details on how they should be protected and also guidelines for creating secure passwords.

A password must:

- never be written down
- never be shared with anyone else
- be unique for a single user account. Don't use the same password if you have multiple accounts, because if one is cracked, they're all cracked!
- never be stored in unencrypted form (i.e. plain text)

The users of a system should be educated as to the risks posed by passwords and the threat that exists from an attacker running a cracking, or password guessing program. Make them aware of the types of words an attacker will be looking for.

Also, the super user, or *root* password must never be revealed to anyone not authorized to use it. A better solution is to implement RBAC or *sudo* to allow administrators a higher privilege without having to provide this password. The *root* password should be kept in a sealed envelope in a secure location so that it can be accessed only in an emergency, and then of course, changed again afterwards.

Recommendations for a Good Password

The following recommendations apply to all user passwords:

- Do not use common dictionary words, or names, or car license plate combinations, phone numbers, social security numbers and so on. Password cracking programs can trawl through millions of potential passwords extremely quickly and are programmed to recognize these kind of patterns

- It has become increasingly common, when choosing a password, to replace some vowels with numerals that are similar in appearance, such as the number "1" for the letters "l" or "i", or "3" for "E". Some password cracking programs look for simple replacements like this
- Do not use common words with a number added, like "89john32" because these are also often included in cracking routines
- DO use a random pattern of numbers and letters, including some letters in UPPER case, but not necessarily at the start of the password
- DO include special characters like "^", ":", "%", "]", "$" and so on as they help to make the password harder to guess
- Do not use a password made up entirely of numbers (Solaris won't actually let you do this)
- Make sure the password is at least 6 characters long. Only the first 8 characters are actually read when a password is entered, so creating a password longer than 8 characters merely adds to the user's problem of trying to remember it
- Use a mnemonic phrase if you know one, and muddle up the UPPER and lower case letters as well as replacing some with numbers, such as "1wLa5Ac", which could be a mnemonic for "I wandered lonely as a cloud"
- Do not use any passwords that have been printed as examples, such as those given in this document because they might be added to a cracker's list.

The items above constitute a defensive password policy in that it is designed to be extremely difficult to crack.

Users with No Password

As a system administrator, you should regularly check for user accounts that have no password assigned. This means that an attacker can login purely be entering the username and pressing <return> and is a huge security risk.

Use the *logins –p* program to report on user accounts with no password set. As an example, the user *nopass* has been configured with no password:

```
# logins -p
nopass          6666    staff           10
```

Password Aging

A password aging policy should be applied to all user accounts, so that a user has to periodically change the password for their account. The period to select depends on the organization and on the security policy that has been implemented, but a common option is to force a change every 28 days, but even this can lead to users becoming fatigued by having to think of a new password every month. Every three or four months will suit some sites more.

Another aspect of password aging is to be able to control how frequently a user may change their own password. One popular scenario is to make a user change the password, only for the user to immediately change it back. For this reason, an option to specify the minimum number of days before a password can be changed is implemented with Solaris 9, as is the number of days' warning a user receives before a password change is required.

The example below shows the command to implement password aging for the user account *temptest*, so that the password must be changed every 90 days, the user can't change it again for 30 days and will receive a warning each day for 10 days prior to the password needing to be changed:

```
# passwd –x 90 –n 30 –w 10 temptest
```

The password aging information is stored in the */etc/shadow* file with the details for each user account.

Password Cracking Tools

There are many password cracking tools available that can easily be downloaded and installed. The system administrator should make use of these tools periodically, with the knowledge of management and users, to test the integrity of passwords. If the system administrator can break the passwords with these tools, then an attacker certainly can too – but the administrator can address weak passwords by locking the affected account until a new password is chosen by the user.

It should be noted that these tools are of limited use to the attacker as it is necessary to have access to the */etc/shadow* file where the encrypted passwords are stored for each user.

TIP: If you are running *NIS*, then the *passwd* map contains the details from both the */etc/passwd* file and the */etc/shadow* file, a well known vulnerability with *NIS*.

The two most popular password cracking utilities are *John The Ripper* and *Crack*.

You can obtain these from:

- John The Ripper – http://www/sunfreeware.com
- Crack – ftp://ftp.cerias.purdue.edu/pub/tools/unix/pwdutils/crack/

For this example, use *John The Ripper* and follow these steps:

- Download the package from www.sunfreeware.com
- Unpack the package using *gzip*
- Install the package using *pkgadd*
- John is installed by default into */usr/local/run*, so go to this directory and prepare the password file by running

```
# ./unshadow /etc/passwd /etc/shadow > passwd.guess
```

- The step above creates the file *passwd.guess*, which *john* will work on to try and obtain the actual password.
- Start the program running. Any passwords that are guessed are, by default, echoed to the screen and also written to an output file in the current directory called *john.pot*.
- At any time during the run, you can press any key to see what the current status of the run is.

The following output shows an actual run:

```
# ./john passwd.guess
Loaded 6 passwords with 5 different salts (Standard DES [32/32 BS])
mysql            (mysql)
12345678         (nopass)
guesses: 2  time: 0:00:00:02 7% (2)  c/s: 26379  trying: chelary -
santand
guesses: 2  time: 0:00:00:27 (3)  c/s: 22615  trying: a9 - tally
guesses: 2  time: 0:00:00:31 (3)  c/s: 22653  trying: dbf - cranda
guesses: 2  time: 0:00:15:43 (3)  c/s: 27337  trying: skaira - mrage
Session aborted
```

Notice the following from the output above:

- Two passwords have so far been guessed, that of users *mysql* and *nopass*
- It only took 2 seconds to guess these weak passwords
- Each status line produced details how many guesses have been successful, the elapsed time so far as well as the current guess range
- The reminaing passwords being attempted are fairly secure because the utility has not been able to easily break them
- Use Ctrl/C to stop the run

Limitations of Password Authentication

A password is only of any use if it is secure. Remember the social engineering tactics mentioned at the start of this document, such as shoulder surfing, posing as a helpdesk engineer and so on – education of users is paramount in being able to successfully defend the passwords in use on a system. If an attacker gains access to the password list, then a cracking tool could be run for days or weeks, thereby increasing the chances of a password being guessed.

All the defensive password techniques are useless if a user then proceeds to write their password down and leave it on a post-it note attached to their desk!

It should also be remembered that password authentication is only one method of gaining access to a system. If the system is not secured in other ways, then an attacker can often gain privileged access without even entering a password.

One popular method of circumventing the password procedure is for an attacker to install a trojaned version of the *login* program. It performs the same function as the legitimate *login* program, but captures the input from a user, i.e. the username and password, then exits (making it look like the user has entered an incorrect password) and then calls the real *login* program. It is easy for a user to be duped by such a program as it appears exactly like the genuine program.

Another more effective method is for an attacker to install a "sniffer" on the network, allowing the capture of packets traveling across the network. In this instance, say, when a user runs *telnet* to connect to a remote host, the password entered will be in clear text and not encrypted – something the sniffer will pick up! The solution to this problem is to always use a secure program, such as SSH (secure shell) for connecting to remote hosts. Using SSH, the traffic between the hosts is always encrypted, preventing a plain text password from being captured by an intruder.

Non-Login Accounts

Solaris 9 makes use of several system accounts that are used as part of the normal running of the Operating environment, these include:

- **daemon**
- **bin**
- **sys**
- **adm**
- **uucp**
- **lp**
- **nobody**

These user accounts are potentially insecure and are rarely checked by administrators to ensure they have not been used. It is advisable to do two things to these accounts:

- Lock each account using **passwd –l**
- Change the login shell to an invalid shell, such as */usr/bin/false* by running **passwd –e** <**username**> and entering a new value

User Security with SU

When a user executes the *su* command, whether it is to the *root* account or any other account, the operation should be logged and controlled.

The file */etc/default/su* achieves this and contains several variables that configure the behavior. The following variables can be set:

- **SULOG** – Normally set to */var/adm/sulog* defines the log file that is written to when the *su* command is run
- **CONSOLE** – Normally commented out, but is set to */dev/console*. If set, this sends a message to the console when *su* is run. It is recommended that this line be uncommented, so the system administrator can monitor its usage
- **PATH** – Normally commented out, but is set to */usr/bin*. This should be set to a minimum number of entries to restrict the commands that can be run
- **SUPATH** – Normally commented out, but is set to */usr/sbin:/usr/bin*. Defines the PATH that is set when the *su* is to *root*. This should be inspected to make sure the current directory "." Is not included (Note that the presence of a trailing ":" character also implies the current directory too)
- **SYSLOG** – Normally set to *YES* so that *su* usage is automatically logged by *syslog*.

Role Based Access Control

RBAC is a tool supplied with the Solaris operating system that provides the facility to give users *root* privileges for a specified command or set of commands, without having to reveal the *root* password.

It provides a fine level of control in that it is fully configurable to suit most requirements. For example, the system administrator wants to delegate backups and cron management to a junior system administrator. This is simple to achieve using roles and profiles within RBAC, the only disadvantage is that the junior system administrator will have a new password to remember.

Additional privileges are achieved through the creation of roles.

A role is a type of user account and is the mechanism by which access is granted to commands using the privileges of another user (normally *root*). There is no direct login to a role, it can only be accessed via the *su* command. Roles are defined in */etc/user_attr* and also has an entry in */etc/passwd*, the same as a normal user account.

A profile is the mechanism where commands can be grouped together to make management and implementation easier. One or more profiles will be associated with a role, and a profile can be associated with multiple roles. Profiles are stored in the file */etc/security/prof_attr*.

An execution attribute contains the actual command to run as well as the user under which it runs. It also associates the profile to which the command belongs. Executions are stored in the file */etc/security/exec_attr*.

A user account is assigned to a role using the *usermod* command and an entry is also added to */etc/user_attr*.

RBAC was covered in detail as part of the Solaris 9 system administrator certification, and is not introduced here.

Creating A Profile

A profile is created by making an entry with an editor, such as *vi*, in the file */etc/security/prof_attr*. To create a new profile for adding user groups add the following entry, noting the number of ":" characters:

```
Group Creation:::Create new groups:
```

Associating Executions with a Profile

The previous action created a profile. At this point the profile does not do anything. The commands for a profile must be entered in */etc/security/exec_attr*. The following example adds the group*add* commands to the "User Creation" profile:

```
Group Creation:suser:cmd:::/usr/sbin/groupadd:euid=0
```

This entry will run the *groupadd* command as user *root* (euid=0).

Creating a Role

Roles are created using the *roladd* command. A role called *newgroup* will be added, which is associated with the "Group Creation" profile:

```
# roleadd -P "Group Creation" -s /usr/bin/pfsh newgroup
```

As a result, the following entry is inserted into */etc/user_attr*:

```
newgroup:::type=role;profiles=Group Creation
```

Note that a role needs to be assigned a default shell that is a profile shell. This ensures that the profile attributes in */etc/security/exec_attr* are used when a command is run. There are three profile shells:

- **/usr/bin/pfsh** – Profile Bourne shell
- **/usr/bin/pfksh** – Profile korn shell
- **/usr/bin/pfcsh** – Profile C shell

Assigning a Role to a User

Having created a role, it needs to be assigned to a user account. To assign the *newgroup* role to the user *temptest*:

```
# usermod -R newgroup temptest
```

The following entry appears in */etc/user_attr* showing the user account *temptest* and that it has been assigned the role *newgroup*:

```
temptest::::type=normal;roles=newgroup
```

Logging in to a Role

To access the functionality of a role, you must first be logged in as a normal user. The user then uses *su* to assume the role identity. For example, user *temptest* assumes the *newgroup* role, running the *id* command before and after:

```
$ id
uid=8888(temptest) gid=10(staff)
$ su newgroup
Password:
$ id
uid=50002(newgroup) gid=1(other)
```

Test the role by checking you can run the required commands as well as normal user commands.

Listing Roles for a User

To list the roles that user *temptest* has been assigned:

```
# roles temptest
newgroup
```

Listing Profiles for a Role

To see the profiles that are associated with the *newgroup* role:

```
# profiles newgroup
Group Creation
Basic Solaris User
All
```

By default, the profiles “Basic Solaris User” and “All” are associated with a role. This allows a user to execute “normal” commands, such as *ls* with the normal user privileges.

Permissions

Whilst it might sound like common sense to most system administrators, file system permissions are frequently overlooked and can, potentially, leave gaping holes for an attacker to exploit. This section looks at the difference between files and directories, in terms of permissions, as well as the risks of having insecure permissions and using the set-uid and set-gid bits.

Directories and Files

The three categories of permission – read, write and execute, have different meanings for files and directories. These are explained below:

- Directories
  - Read – This allows the directory to be read, but the files cannot be listed
  - Write – This allows files to be created, renamed and deleted, regardless of the individual permissions set on a file within the directory
  - Execute – This allows the directory to be listed
- Files
  - Read – This allows a file to be read and copied
  - Write – This allows a file to be written, but it should be noted that this permission alone does not allow read access as well, nor does it allow a file to be deleted
  - Execute – A compiled program can be executed, but a shell script can only be executed if read permission is also granted

The Set-Uid and Set-Gid Permissions

In addition to the three categories of permission mentioned above, there are two further permissions that possess the ability to cause serious damage if they are not used with extreme caution. The set-uid permission, when set on a program or shell script, will assume the owner's privileges when executed. Similarly, the set-gid permission will assume the group owner's privileges.

To implement the set-uid permission, use the *chmod* command, in either symbolic or absolute mode. The following example shows how to apply the set-uid permission to the script, *test.sh*, which already has a permission of *755* (or *rwxr-xr-x*):

```
# chmod u+s test.sh
# chmod 4755 test.sh
```

Both of the above commands achieve the same thing.

To implement the set-gid permission for the same script:

```
# chmod g+s test.sh
# chmod 2755 test.sh
```

Implications of Lax File and Directory Permissions

The following list contains the potential risks of bad permission management:

- An attacker can install Trojan files easily and gain full control of the system
- The data is open to theft or malicious tampering
- Confidentiality and integrity of the data is compromised

- An attacker can gain valuable information about the system which can be used later to aid further attacks
- Files can be accidentally deleted or corrupted by legitimate users
- Sensitive management information could potentially be read by employees
- Customer confidence in the organization can suffer greatly if data is exposed in the public domain
- An organization might be vulnerable to prosecution if data protection legislation is deemed to have been breached through mis-management of customer information held on a system with lax permissions

Permissions can be made more secure by applying a more restrictive *umask* on a system wide basis. The standard *umask* of *022* only restricts "group" and "other" categories from writing. If a *umask* of *027* is implemented, then the "other" category will have no access at all, by default.

Access Control Lists (ACL)

Access Control Lists provide a much finer level of control over file permissions. The standard Unix permissions are sometimes not enough to do what you want, so ACLs allow permissions to be set on a per-user basis or allow other groups access to files without having to give access to everybody.

ACL entries are applied to files using:

- **setfacl** – to establish, modify or delete an ACL – as well as an option recalculate the ACL mask
- **getfacl** – to list the details of an ACL

Identifying an ACL

The simplest method of identifying that an ACL has been placed on a file is run a long listing, using *ls*.

The following two listings show the file *testfile* before and after an ACL has been applied to the file:

```
# ls -l testfile
-rwxr-----    1 johnp    johnp        252 Apr 12 15:41 testfile

# ls -l testfile
-rwxr-----+  1 johnp    johnp        252 Apr 12 15:43 testfile
```

Notice the addition of a "+" character. It is this that indicates an ACL is present.

Setting ACLs

To set

```
# setfacl -s user::rwx,g::r--,o:---,mask:rw-,u:temptest:r-- testfile
```

To see the ACL just created, use the *getfacl* command:

```
# getfacl testfile
```

```
# file: testfile
# owner: john
# group: john
user::rwx
user:temptest:r--               #effective:r--
group::r--                      #effective:r--
mask:rw-
other:---
```

The ACL allows the user *temptest* to have read access to the file *testfile*.

Note that if you run *setfacl –s* on an existing ACL, it will replace the entire ACL, overriding the current ACL.

Modifying an ACL

If the ACL above were to be modified so that the *mask* setting was set to ---, then the user *temptest* would no longer have access to the file even though the ACL indicates that read access was granted. This is because the *mask* setting identifies the maximum access that can be granted, overriding the individual permissions:

```
# setfacl –m mask:--- testfile
```

Now the ACL looks like this:

```
# getfacl testfile
```

```
# file: testfile
# owner: john
# group: john
user::rwx
user:temptest:r--               #effective:---
group::r--                      #effective:---
mask:---
other:---
```

Deleting an ACL

To remove an ACL, use the *setfacl –d* command to remove the specific permissions. When the last permission is removed, there is no longer an ACL on the file:

```
# setfacl –d u:temptest testfile
```

Recalculating the Mask of an ACL

The mask of an ACL reports on the effective permissions that are in effect on an ACL. When the ACL permissions are modified, the permissions need to be recalculated based on the mask:

To reapply the previous modification with the mask recalculation:

```
# setfacl –r –m mask:--- testfile
```

PAM and Kerberos

This section describes two methods of improving the authentication mechanism on a Solaris 9 system.

Pluggable Authentication Module (PAM)

PAM is a framework that provides the facility to add new authentication techniques without having to make changes to system services. New modules can simply be "plugged in" to integrate with the existing system.

PAM is used primarily for authentication with programs like *login*, *telnet*, *ftp*, *rlogin* and so on. It is configured using the file */etc/pam.conf*. The relevant lines for the *login* program in the default *pam.conf* are shown here:

```
login   auth requisite          pam_authtok_get.so.1
login   auth required           pam_dhkeys.so.1
login   auth required           pam_unix_auth.so.1
login   auth required           pam_dial_auth.so.1
```

The format of the file is:

- Service – The program being authenticated, such as *login*. A service called *other* can also be used to make management of a number of services easier
- Module Type – The type of service being provided – can be *auth*, *account*, *session* or *password*

- Control Flag – The deciding factor on what constitutes a success or failure – can be *requisite*, *required*, *optional* or *sufficient*. When an *auth* module is used for example, the controls function like this:
  - Requisite – The module being executed must be successful for any further authentication to be allowed.
  - Required – The overall result of the authentication must be successful. If a failure occurs in a module, all others are still tried, but an error is returned
  - Optional – This flag means that if a failure occurs in a module, then the overall result can still be successful, if another module returns a successful completion
  - Sufficient – As long as this module is successful, then there is no need to run any others – the authentication can finish and return successful
- Module Path – The pathname to the module
- Module Option(s) – Specific options that can be passed to the module, such as *debug* or *use_first_pass* (this latter option allows the password entered by the user to be automatically inserted in subsequent authentication modules, eliminating the need for the user to enter the password multiple times)

Deploying PAM in a Production Environment

Before using PAM in a live environment, consider the following aspects:

- Choose the control flags carefully to ensure that the right level of security is being applied. This is particularly relevant when deciding to use the *sufficient* or *optional* flags
- Decide which modules you need to use to obtain the required level of security
- Pay special attention to the services being used and highlight any that might need additional authentication modules for added security
- Don't apply unnecessary levels of security, they only serve to add to the complexity and the overhead required
- Select the order in which modules should be used. If a failure will stop the entire authentication process, then put this one above other, less important modules so that unnecessary processing is avoided

Add a new PAM Module

Follow these steps when adding a new PAM module:

- Login and become superuser (*root*)
- Ensure you have selected the type of authentication required as well as any options that might be needed
- Install the new module in */usr/lib/security*

- Make sure the module is owned by *root* and the permissions should be equal to *555* (or *r-xr-xr-x*). You should note that the default installation puts permissions at *755*, so you might want to change these
- Edit the PAM configuration file, */etc/pam.conf* and add the new module to the services it is going to provide authentication for
- It is always advisable to reboot the system and then test the new module to ensure it is working as expected. A reboot is not mandatory, but is good practice

For more information on PAM, see http://docs.sun.com and goto the System Administration Guide:Security Services manual for Solaris 9.

Kerberos / SEAM

SEAM (Sun Enterprise Authentication Mechanism) is a single sign-on utility that is based on the Kerberos version 5 security ticketing system.

Kerberos basically works on a system of granting tickets which provide access to systems or applications. It is a client/server based service that handles authentication across a network.

How Kerberos Works

The following steps explain how Kerberos functions to provide NFS access to a client:

- A client wants to access an NFS file system and requests a “ticket granting ticket” or TGT from a “key distribution center” or KDC. A KDC is a server that authenticates the client and issues the TGT
- The client uses its own password to decrypt the TGT, thereby proving the identity to be authentic
- Having obtained a TGT, the client can now request tickets to access the NFS server that is sharing the required file system
- The client issues a request for the NFS access to the KDC and also sends its TGT as proof of identity
- The KDC checks the TGT for authenticity and then issues a ticket for access to the NFS server
- The client, now in possession of a ticket to access NFS, sends the ticket to the NFS server
- The NFS server allow the client to access its resources

Limitations of Kerberos

Even though Kerberos is a fairly robust solution to network authentication, it does have the following limitations:

- Kerberos is not a transparent service, like PAM where modules can be plugged in. In order to use Kerberos, each service needs to be modified
- The KDC server provides a single point of failure and could potentially stop all access to services if it becomes unavailable
- Kerberos stores all of its encrypted passwords using a single key, so if the server is ever compromised, then ALL passwords must be changed
- The KDC server must be extremely secure and "locked down" because it would be a higher priority target for an attacker
- Kerberos stores its tickets in the */tmp* directory, so is not geared towards multi-user systems. It makes the tickets vulnerable to theft and spoofing of services

Host and Network Prevention

This section is concerned with securing access to the network or system. It describes some basic terms that you need to be familiar with and shows how to manually restrict the services and functions that the system is running. Also, using the Solaris Security Toolkit is described, which allows a system to be secured automatically.

Fundamentals

This section describes some basic terms used in conjunction with network and host security.

Firewall

Derived from the firefighting technique of building a barrier to prevent a fire from spreading. A firewall is a suite of programs that protects the assets of a private network from assets and users from other networks. It is usually located at or near the gateway to a company, on the external interface (i.e. one facing the Internet). Often these are dedicated appliance systems, like the CISCO PIX firewall, but can also be software run on a workstation or server, like Sun Microsystems Sunscreen firewall. Examples of firewall functionality include:

- **Packet Filtering** – one which inspects each packet and checks the source and destination address for validity
- **Stateful Application Filtering** – one which inspects each packet and decides its validity based on the actual content of the packet as well as the source and destination addresses. This type of firewall is much more secure, but requires greater resource to process the overhead involved and is more likely to affect network response times
- **Proxy** – where the real address of a host is hidden, or masked, from the outside world. The proxy function forwards packets onto the "real" internal host

- **Network Address Translation (NAT)** – where a corporate network can be made to look (externally) like it has only one address, or a limited number of addresses. Numerous internal addresses can be mapped to a single external IP address, protecting the identity of the internal hosts.

A firewall works on a set of rules which either allow or deny certain addresses or types of data. The rules are usually processed in a top-down fashion, stopping when a match is found. It is always good practice to insert a deny rule for all network traffic at the end of the ruleset to catch any packets that don't match any other rules.

IPsec

Internet Protocol security (IPsec) is a framework for applying security at the network transport level, instead of at the application level like a lot of other security mechanisms. Two main services operate here:

- **Authentication Headers (AH)** – where the sender must be authenticated before data packets will be allowed
- **Encapsulating Security Payload (ESP)** – where the sender is authenticated, but the data can also be encrypted for added protection

Network Intrusion

A network intrusion is said to have occurred when there has been unauthorized access to the network. This can take the form of a port scanning operation, where a potential attacker might be trying to find vulnerabilities in your network, or

Intrusion Detection

The activity of spotting an attempted intrusion on your network. An intrusion can often be identified by the type of activity being performed by a potential attacker, lots of packets being sent to different ports for example. Intrusion Detection Systems (IDS) such as *Courtney*, *Gabriel* and *snort* look specifically for these kind of patterns and alert the administrator to any suspicious activity. It should be noted that an IDS cannot prevent an attack, or intrusion, from taking place, it can only detect it.

Restricting Network Services

Inetd Services

Network services are controlled via the file */etc/inetd.conf* and are implemented using the *inetd* daemon. To restrict services, edit */etc/inetd.conf* and place a "#" in column 1. This makes the whole line a comment and is ignored by the *inetd* daemon.

It is good practice to disable all services and then only re-instate the services that are genuinely necessary. When the file has been edited, *inetd* must be instructed to re-read the configuration file, so that changes are made operational. This can be easily done using the following command:

```
# pkill –HUP inetd
```

The command above sends a "Hangup" signal to the daemon causing it to examine its configuration file again. You could also reboot the system or find the *pid* of *inetd* and restart it manually, using *kill –1*. (Note that *–1* is the same as *HUP*).

Run Control Services

Disable any run control scripts that are not needed. These scripts are found in the */etc/rc*.d/* directories. Any services not required should be renamed. A good practice is to rename the script so that it is preceded by an underscore character (_) or by changing the initial letter to lowercase. The following example disables the *dtlogin* script to start the graphical login utility:

```
# mv /etc/rc2.d/S99dtlogin /etc/rc2.d/_S99dtlogin
```

Now, when the system reboots, this script will be ignored, because startup scripts must being with an uppercase "S" character and shutdown scripts must begin with an uppercase "K" character.

Remote Procedure Call (RPC) Services

If you don't need to use RPC at all, then turn it off by disabling the following script:

```
# mv /etc/rc2.d/S71rpc /etc/rc2.d/_S71rpc
```

If you do need to use RPC, then restrict the programs used. There are a number of RPC programs in the file */etc/inetd.conf*. Normally these can all be disabled. Check the entries in the file */etc/rpc* and comment out those that are not required. You can then restart the *rpcbind* daemon to implement the changes.

Host Hardening

Host hardening is the action of making a system more secure. The more hardening that is done on a system, the more secure it will be against external (and internal) attack. The act of hardening involves removing potential vulnerabilities and security risks that are present in the Solaris operating system, but most of the techniques used here are simply good practice for securing your systems. The following techniques are some that are recommended to improve the security of your system:

- Install Solaris from recognized media
- Perform an Initial installation so that any residual information or files is cleared

- Only install the Solaris cluster containing packages that you actually need. There is no need to install everything if it's not required and it will created unnecessary security risks if you do
- Restrict network services in */etc/inetd.conf*
- Restrict RPC services
- Manage user accounts effectively by including expiry dates and locking the passwords of dormant accounts
- Secure system accounts such as *adm*, *lp*, *sys*, *nobody* etc. These accounts have no password by default, so lock them using *passwd –l*
- Remove NFS software if your system is not going to act as an NFS server or client
- Secure the system console at both the OpenBoot PROM level and also restrict *root* logins to only be allowed from the console itself and not remotely
- Mount filesystems read-only where appropriate and include the *nosuid* flag so that any programs or file with set-uid or set-gid privileges are negated
- Review all set-uid and set-gid programs and scripts. These are potentially dangerous and could compromise the security of the system
- Restrict *cron*, *at* and *batch* actions to prohibit automatic processing by unauthorized users
- Implement roles using RBAC to give additional privileges to users without having to allow *root* access
- Modify the default *umask* value, normally set to *022*. Set a new default of *027* for example so that other users have no access to files and directories
- Enable logging and accounting
- Display suitable access warnings in the appropriate files where users can access your system remotely
- Disable the automounter if this facility is not to be used. Rename the startup script */etc/rc2.d/S74autofs* to do this

See http://www.sun.com/solutions/blueprints/1202/816-5242.pdf for full details of how to implement the techniques above.

Solaris Security Toolkit

The Solaris Security Toolkit (SST), also known as the Jumpstart Architecture and Security Scripts Toolkit (JASS) provides an easy, automated method of securing your system. The package can be downloaded from Sun Microsystems at:

http://wwws.sun.com/software/security/jass/index.html

and Sun blueprints for a quick-start guide can be found at:

http://www.sun.com/solutions/blueprints/0601/jass_quick_start-v03.pdf

and for a full install, configure and run guide go to:

http://www.sun.com/solutions/blueprints/0601/jass_conf_install-v03.pdf

SST installs by default into */opt/SUNWjass* and the current version at the time of writing is 4.0.1.

SST can be implemented in standalone mode or as part of a JumpStart configuration. The advantage of using SST with JumpStart is that systems can be automatically hardened as part of the installation process.

Installing SST

To install SST, you must first download the software from the Sun website. Choose the package option for the easiest install, which is delivered as a compressed file called:

SUNWjass-4.0.1.pkg.Z

Uncompress the file using the following command:

```
# uncompress SUNWjass-4.0.1.pkg.Z
```

which will leave the package file: SUNWjass-4.0.1.pkg

Now using *pkgadd*, install the software:

```
# pkgadd –d ./SUNWjass-4.0.1.pkg
```

Configuring SST

There are several directories in the SST which serve a different purpose. The most important ones are listed here:

- Drivers – This contains the master scripts to determine which other scripts are called, from the *Finish* directory and which files are to be replaced
- Files – This contains the files that will be replaced during the run
- OS – This contains Solaris operating system images for use with the JumpStart installation process
- Packages – This contains any additional software packages to be installed as part of the run. This can include application software
- Patches – This contains recommended and security patch clusters, which should be extracted into this directory. The patches will be applied as part of the jass run
- Profiles – This contains JumpStart profiles for use when installing using this method. A profile includes such items as the cluster to install, the disk layout. One or more of the profiles must be listed in the *rules* file, just like any other JumpStart profile
- Finish – This contains the scripts that will carry out the modifications and changes to the system

- Audit – This contains the scripts to run in order to carry out a verification check jass run. These scripts do not make changes, they just analyze the current state of the system and report the vulnerabilities it finds

The *secure.driver* script is shown here:

```
# cat secure.driver
#!/bin/sh
#
# Copyright (c) 2000-2003 by Sun Microsystems, Inc.
# All rights reserved.
#
#ident  "@(#)secure.driver      3.6     03/11/24        SMI"
#
# The purpose of this driver is to act as a wrapper calling the
# "configuration" and "hardening" drivers.
#
DIR="`/bin/dirname $0`"
export DIR
. ${DIR}/driver.init
. ${DIR}/config.driver
. ${DIR}/hardening.driver
```

Running SST

The most common option to run the SST is to use the standalone method, which executes the *secure.driver* file. Start the SST run by entering the command:

```
# ./jass-execute -d secure.driver

[NOTE] Executing driver, secure.driver
=======================================================================
secure.driver: Driver started.
=======================================================================
JASS Version:  4.0.1
Node name:     ultra
Host ID:       8f88ffe0
Host address:  192.168.1.1
MAC address:   8:0:20:00:00:00
OS version:    5.9
Date:          Sun Apr 11 17:50:14 BST 2004
=======================================================================
secure.driver: Finish script: install-templates.fin
=======================================================================
Copying personalized files.
[NOTE] Copying /.cshrc from /opt/SUNWjass/Files/.cshrc.
```

```
[NOTE] Copying /.profile from /opt/SUNWjass/Files/.profile.
========================================================================
secure.driver: Finish script: print-jass-environment.fin
========================================================================
   JASS_ACCT_DISABLE
   daemon bin adm lp uucp nuucp nobody smtp listen noaccess nobody4
smmsp
   JASS_ACCT_REMOVE
   smtp listen nobody4
   JASS_AGING_MINWEEKS
[output truncated]
```

Updating an SST Run

SST should be run periodically to ensure that no changes made to the system have invalidated any of the security modifications made when it was first run. This is definitely true if patches have subsequently been installed which could undo some of the changes.

To update SST, simply run it again, or if you have created a modified script file to run, then execute the modified one to install your own specific modifications.

Undoing SST

Each invocation of *jass-execute* creates a log of the run in the directory:

```
/var/opt/SUNWjass/run
```

To undo SST, enter the following command from the JASS directory and select the run, based on the fully qualified date and time file that represents the jass run you want to undo:

```
# ./jass-execute –u
```

```
[NOTE] Executing driver, undo.driver
Please select a JASS run to restore through:
1.  April 11, 2004 at 17:50:13 (//var/opt/SUNWjass/run/20040411175013)
Choice ('q' to exit)?  1
[NOTE] Restoring to previous run /var/opt/SUNWjass/run/20040411175013
====================================================
undo.driver: Driver started.
[output truncated]
```

This option is especially useful if you have installed patches, updated SST and then experienced problems because you are able to backout only the latest changes, leaving all previous modifications intact.

You should note that not all SST actions can be undone, only those that are called by a script. This needs to be borne in mind when trying to undo SST, because you might get some unexpected results and residual security implementations left over.

Verifying SST

Earlier release of SST called this an *Audit* run, but it is now known as a verify run. This option does not actually make any changes, but runs checks against the items that would be changed so it can identify what needs to be done. Each check the run makes is marked either "PASS" or "FAIL", a "PASS" indicating that the security feature is already implemented. The two sets of output below show partial results for the same checks, the first one before SST was run and the second one after it has been run and the system rebooted:

Before:

```
# Checking the nscd time-to-live parameters.

[FAIL] positive-time-to-live for 'passwd' is not '0'.
[FAIL] negative-time-to-live for 'passwd' is not '0'.
[FAIL] positive-time-to-live for 'group' is not '0'.
[FAIL] negative-time-to-live for 'group' is not '0'.
[FAIL] positive-time-to-live for 'hosts' is not '0'.
[FAIL] negative-time-to-live for 'hosts' is not '0'.
[FAIL] positive-time-to-live for 'ipnodes' is not '0'.
[FAIL] negative-time-to-live for 'ipnodes' is not '0'.
```

After:

```
# Checking the nscd time-to-live parameters.

[PASS] positive-time-to-live for 'passwd' is '0'.
[PASS] negative-time-to-live for 'passwd' is '0'.
[PASS] positive-time-to-live for 'group' is '0'.
[PASS] negative-time-to-live for 'group' is '0'.
[PASS] positive-time-to-live for 'hosts' is '0'.
[PASS] negative-time-to-live for 'hosts' is '0'.
[PASS] positive-time-to-live for 'ipnodes' is '0'.
[PASS] negative-time-to-live for 'ipnodes' is '0'.
```

At the end of the verify run, the total number of failures is indicated. After the SST run, nearly all of these will relate to packages being installed. You can remove these packages using the *pkgrm* command if they are not going to be used.

Network Connection Access, Authentication and Encryption

The final section looks at remote connections and the basics of cryptology.

TCP Wrappers

TCP Wrappers provides additional logging and authentication for the network daemon processes such as:

- **ftp**
- **telnet**
- **rlogin**
- **rsh**
- **tftp**
- **exec**
- **finger**

The wrappers are small daemon programs that "wrap" the actual network daemons, like *in.telnetd*.

You should note that TCP Wrappers doesn't implement a full security mechanism, but does offer greater protection than the standard network daemons.

If you've followed the advice earlier in this document and disabled the network services in */etc/inetd.conf*, then there is no need for TCP Wrappers because you are not using any of these services.

There are two methods of using TCP Wrappers, hidden and visible. This document concentrates on visible wrappers.

Hidden TCP Wrappers requires modification to all of the network daemons and is much more complicated when it comes to upgrading the operating system, unlike visible TCP Wrappers which would only require the modification of one file, namely */etc/inetd.conf*, if the system was to be upgraded.

TCP Wrappers can be downloaded from:

http://www.sunfreeware.com and installs by default into */usr/local*.

Configuring TCP Wrappers

To configure TCP Wrappers, simply modify the relevant service line in */etc/inetd.conf*, so to install this facility on the *telnet* service, change this line:

```
telnet  stream  tcp6  nowait  root  /usr/sbin/in.telnetd  in.telnetd
```

with

```
telnet  stream  tcp  nowait  root  /usr/local/bin/tcpd  in.telnetd -d1
```

Denying and Allowing Host Connects

The files */etc/hosts.allow* and */etc/hosts.deny* can be created to allow or deny specific connections.

Note that if there is no entry, then access is allowed.

One way round this is to edit the two files like this:

In */etc/hosts.deny* put the following entry:

ALL: ALL:

Then, to allow for example, *192.168.1.1* to use *telnet*, put the following entry in */etc/hosts.allow*:

in.telnetd: 192.168.1.1

The solution above will only allow *192.168.1.1* to connect. All other connections will be refused.

Denying Connections with a Banner Message

When a connection is refused, it is good practice to supply an information message. TCP Wrappers allows this by creating a number of banner files that can be displayed to the user when a connection is refused. Follow these steps to create a standard banner message for the connection daemons:

- Create the directory */etc/tcpd.deny*

```
# mkdir /etc/tcpd.deny
```

- Copy the *Makefile* to this directory

cp /usr/local/doc/tcp_wrappers/Banners.Makefile /etc/tcpd.deny/makefile

- Create and edit the file *prototype* in */etc/tcpd.deny*. An example message is shown below:

```
Warning! This is an unauthorized connection and has been logged.
This host is constantly monitored and violations will be reported.
```

- Save the file and change to the directory, then run *make*.

```
# cd /etc/tcpd.deny
# make
cp prototype in.telnetd
chmod 644 in.telnetd
sed 's/^/220-/' prototype > in.ftpd
chmod 644 in.ftpd
echo 'main() { write(1,"",1); return(0); }' >nul.c
gcc  -s -o nul nul.c
```

```
rm -f nul.c
( ./nul ; cat prototype ) > in.rlogind
chmod 644 in.rlogind
```

- This creates banner files for *in.ftpd*, *in.telnetd* and *in.rlogind*.
- Now when an unauthorized host tries to connect, the banner message will be displayed and the connection refused.

Logging

TCP Wrappers writes to the log files using *syslog*.

Valid connect messages are written to the *auth.info* level and refused connections are written to the *auth.warning* level. You can reconfigure *syslog* by editing */etc/syslog.conf* if you want these messages written to a separate log file. An example of each kind of message is displayed below:

```
Apr 10 13:23:03 ultra10 in.telnetd[600]: [ID 947420 mail.warning]
refused connect from ultra1.mobileventures.homeip.net

Apr 10 13:28:07 ultra10 in.telnetd[603]: [ID 927837 mail.info] connect
from ultra2.mobileventures.homeip.net
```

Validating TCP Wrappers

Use the *tcpdchk* command to check the configuration of TCP Wrappers.

```
# /usr/local/bin/tcpdchk -av

Using network configuration file: /etc/inet/inetd.conf

>>> Rule /etc/hosts.allow line 1:
daemons:  in.telnetd
clients:  192.168.123.1
warning: /etc/hosts.allow, line 1: implicit "allow" at end of rule
access:   granted

>>> Rule /etc/hosts.deny line 1:
daemons:  ALL
clients:  ALL
option:   banners /etc/tcpd.deny
access:   denied
```

You can also check an individual host to see whether it is permitted to connect using a specific service. For example to check if host *ultra1* can use *telnet* to your system, run:

```
# /usr/local/bin/tcpdmatch in.telnetd ultra1
```

```
warning: ultra1: hostname alias
warning: (official name: ultra1.mobileventures.homeip.net)
client:   hostname ultra1.mobileventures.homeip.net
client:   address  192.168.1.1
server:   process  in.telnetd
matched:  /etc/hosts.allow line 1
access:   granted
```

Cryptology

Terminology

This section describes a number of terms used in cryptology:

- **Secret-key** – Also known as *private-key and symmetric key*. It describes a method by which date is encrypted and decrypted using the same key. This method is less secure than the public-key method because there is a vulnerability when the key is distributed to other systems that need to send or receive secure data. If an attacker obtains this key, then the data can easily be converted to plain text and read
- **Public-key** – Also known as *public and private key pairs* and *asymmetric keys*. It describes a more secure method where two keys work in partnership to send and receive secure data. One key is used to encrypt the data (a private key, which is only held by the sender and a public key, which is used to decrypt the data that is received. Because the private key is kept secret by the sender, the receiver of data can authenticate its origin. Only the public key is distributed to other hosts that need to receive secure data.
- **Hash Function** – Also known as *hash algorithms*, these provide the mechanism for encrypting data and checking its integrity. Popular hash functions include the Message Digest algorithm number 5 (MD5), a 128-bit algorithm and the Secure Hash Algorithm (SHA-1), a 160-bit algorithm. Hash functions can be used to detect whether data has changed during transit, whether from corruption on the network or maliciously.
- **Encryption** – This ensures that data in transit cannot be read by an attacker, even if access to the data itself is obtained. You should note that encryption only applies to data whilst it is in transit.
- **Authentication** – The action of reliably determining the sender's or receiver's identity.

Solaris Secure Shell

The Secure Shell (SSH) is delivered as part of the standard Solaris 9 implementation and provides secure network connectivity between hosts, replacing insecure alternatives like *ftp*, *telnet* and *rcp*.

SSH comes with a number of tools:

- **sftp** – Secure ftp
- **sftp-server** – Secure ftp server

- **ssh** – Secure session connection to replace *telnet*
- **scp** – Secure copy of files between hosts
- **sshd** – The server daemon that processes requests from clients
- **ssh-agent** – The authentication agent that holds the "keys"
- **ssh-add** – This registers new keys with the agent
- **ssh-keygen** – Used to create a new pair of keys for the client and server authentication

Configuring the Server

The SSH server uses the configuration file */etc/ssh/sshd_config*. In this file you can configure such aspects as:

- The SSH protocols to use (1 or 2, or both)
- The port to listen on (normally 22), but a nonstandard port could be configure here
- The location for the storage of the keys
- Allow or disable X11 port forwarding
- Allow or disable other forms of authentication (such as *.rhosts*)

Starting and Stopping SSHD

The ssh server (sshd) is started and stopped via a startup script in */etc/rc3.d* called *S89sshd*, which is a link to */etc/init.d/sshd*.

Configuring the Client

The client is configured using the configuration file */etc/ssh/ssh_config*. You can configure the following options:

- The type of authentication used
- The port to be used for ssh to communicate
- The location of the files to store client keys
- The encryption algorithm to use. This is determined by the client not the server
- Configure specific host options. For example, some hosts could be configured to communicate on different ports
- Prevent access. Normally a host that is not known produces a warning, but unknown connections can be prevented instead

Generating a Client Key

The client generates a key pair (private and public keys) by using the *ssh-keygen* command. When this command is run, the client has to enter a passphrase that is to be used to create the keys. An example is shown below:

```
$ ssh-keygen

Enter file in which to save the key(/export/home/john/.ssh/id_rsa):
Generating public/private rsa key pair.
Enter passphrase(empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /export/home/john/.ssh/id_rsa.
Your public key has been saved in /export/home/john/.ssh/id_rsa.pub.
The key fingerprint is:
md5 1024 61:e5:81:a3:4d:81:aa:41:f4:ae:c4:89:02:8c:88:e4 john@ultra
```

Using ssh

Make a secure connection to another host, instead of a *telnet* connection by using the *ssh* command:

```
$ ssh john@ultra.mobileventures.homeip.net

Enter passphrase for key '/export/home/john/.ssh/id_dsa':
John@ultra.mobileventures.homeip.net's password:
Last login: Sat Apr 10 15:36:13 2004 from 192.168.1.3
$
```

Note that when strict authentication is in use, you will have to enter not only your SSH passphrase, but if that is successful, also the password to actually login.